

3

Selectors

In this chapter, you learn about the different types of selectors that CSS supports. In Chapter 2, you learned about the type selector, which is a selector that applies style sheet declarations by using the HTML element's name. "Selectors" is an area of CSS that I discuss that has spotty support with regards to IE 6. To those ends, as I introduce each section, if a selector is not supported by IE 6, I note that. IE 7 features much better selector support, and in fact supports nearly all of the selectors discussed in this chapter, but there are a few selectors that IE 7 doesn't support. This is also noted where appropriate. Other browsers such as Mozilla Firefox, Safari, and Opera all have excellent support for the selectors discussed in this chapter. With each example, I note what browser you should use to view the example, and which browsers the example won't work with.

You may wonder why I bother discussing selectors that don't work in IE 6. I chose to include the selectors with at least some browser support, because each reader's needs and development requirements are different. If you are, for instance, developing a corporate intranet-based application in which you have full control over the browser the end user is using, your needs are different from someone who is developing a publicly-accessible Internet website. Someone developing a corporate intranet site can, for instance, choose Mozilla Firefox as their development platform, rather than IE 6, or that corporation may choose to upgrade to IE 7. In short, not everyone has the same end-user requirements for browser usage, and this book is written with that in mind.

You can also use JavaScript applications that enable a greater spectrum of CSS support in IE 6. *JavaScript* is a programming language that you can use to create scripts that are included in an HTML document in much the same way as CSS. JavaScript opens up possibilities that HTML and CSS alone aren't capable of. Using JavaScript technology, you can give IE 6 CSS capabilities that are impossible without it. When you use JavaScript, most of the very same examples that you encounter in this chapter that don't work in IE 6, can work in IE 6 flawlessly and reliably, and without the end user having to upgrade IE 6 or take any other action. I discuss how you, too, can harness this incredibly useful, and seemingly magical, technology in Chapter 16, available at www.wrox.com/go/beginning_css2e. The best part is, you need no experience programming JavaScript to use the technology that I present in Chapter 15. So, if you feel discouraged by IE 6's lack of support for many of these useful selectors, continue on; there are hackadelic methods yet to be discussed.

Part I: The Basics

You may also wish to see more practical applications of the features presented in this chapter. If that is the case, I provide some real-world projects at the end of this book that help you to put CSS into a real-world context. As is the case throughout this book, I present all the bits and pieces of the language with proof-of-concept examples, and then later in the book you see how to put it all together with some real, skill-building projects. Alternatively, you may also be interested in my book *CSS Instant Results* (Wrox, 2006), an intermediate-level CSS book that focuses on real-world projects exclusively.

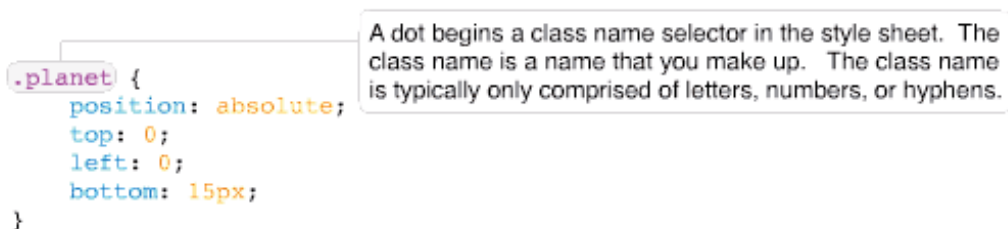
I begin the discussion of selectors with the most common and widely supported selectors, class and id selectors.

Class and ID Selectors

Class and id selectors are the most widely supported selectors. In fact, they are as widely supported as the type selector introduced in Chapter 2. There are two types of selectors. The `class` attribute is more generic, meaning it may encompass many elements in a given document, even elements of different types or purposes. On the other hand, you can use the `id` attribute only once per document. The name `id` tells you that the id must be unique. Besides using it in CSS, you can also use an element's `id` to access it via a scripting language like JavaScript. You can also link to the location of the element with an `id` name using anchors. Anchors are appended to URLs to force a browser to go to a specific place in a document. So the `id` attribute serves more than one purpose. Think of it as an element's address inside a document — no two addresses can be the same. The discussion continues with class selectors.

Class Selectors

Figure 3-1 is an example of a class name selector.



```
.planet {  
  position: absolute;  
  top: 0;  
  left: 0;  
  bottom: 15px;  
}
```

Figure 3-1

The class name selector begins with a dot, followed by the class name itself, which you choose. Typically, the class name is comprised of letters, numbers, and hyphens only, since this provides the best compatibility with older browsers. Class names also cannot include spaces. In Figure 3-2, you see the element that the class name `planet` applies style to in the HTML document.

```
<div class='planet'>
  <img src='images/jupiter.png' alt='Jupiter' />
</div>
```

The class name is included in the HTML document using the `class` attribute.

Figure 3-2

The dot appearing before the class name in the CSS rule tells CSS that you are referencing a class selector. The dot does not need to appear in the `class` attribute value itself; in fact it cannot, because the value of the `class` attribute is just the class name itself.

When used in this context, the type of element doesn't matter. In other words, you can also apply the class to other elements, as is illustrated in Figure 3-3.

```
<div class='planet'>
  <img src='images/jupiter.png' alt='Jupiter' class='planet' />
</div>
```

Class names don't have to be specific to a type of element. The `<div>` element and the `` element can both have a class name of *planet*, if desired.

Figure 3-3

The same rule applies to the `` element as applies to the `<div>` element. Both now have an absolute position, offset from the top zero pixels, offset from the left of zero pixels, and offset from the bottom of 15 pixels. What if you wanted to give both the `<div>` and `` element the same class name, but have a style sheet rule that applies to `<div>` elements, but not `` elements? You can do that, too. Limiting a class selector to a type of element is demonstrated in Figure 3-4.

```
div.planet {
  position: absolute;
  top: 0;
  left: 0;
  bottom: 15px;
}
```

Append a class selector with a type selector to limit the style sheet rule to a certain type of element.

Figure 3-4

In Figure 3-4, you see the combination of two types of selectors that you are already familiar with, the type selector from Chapter 2, and the class selector. When you append a type selector to a class selector, you limit the scope of the style sheet rule to only that type of element. In Figure 3-4, the rule is limited so that it only applies to `<div>` elements, causing it to no longer apply to `` elements, or any other type of element for that matter. You can still create additional rules that reference other elements, such as a new rule that only applies to `` elements with a class name of *planet*, such as `img.planet`, but the rule that you see in Figure 3-4 applies exclusively to `<div>` elements with a class name of *planet*.

Part I: The Basics

Elements can also be assigned more than one class name. Figure 3-5 shows an example of this.

```
<div class='planet jupiter'>
  <img src='images/jupiter.png' alt='Jupiter' class='planet' />
</div>
```

Multiple class names can be applied to a single element. Each class name must be separated by a single space.

Figure 3-5

The value of this `class` attribute actually contains two class names: `planet` and `jupiter`. Each class name in the attribute is separated by a space. In the corresponding style sheet, the two classes may be referenced by two separate rules, as illustrated in Figure 3-6.

```
div.planet {
  position: absolute;
  top: 0;
  bottom: 15px;
}
div.jupiter {
  left: 0;
}
```

The `<div>` with both `planet` and `jupiter` class names can be referenced by two separate rules.

Figure 3-6

The two style sheet rules in Figure 3-6 result in the `<div>` element with both `planet` and `jupiter` class names receiving the declarations of both rules.

The class names may also be chained together in the style sheet, as shown in Figure 3-7.

```
div.planet.jupiter {
  left: 0;
}
```

Class names can be chained to one another to reference an element that has multiple class name values.

Figure 3-7

The preceding rule applies only to elements that reference both class names in their `class` attribute.

IE 6 interprets chained class names per the CSS 1 specification, which did not allow chained class names in the style sheet. In IE 6, only the last class name in the chain is recognized. In the preceding example, IE 6 would interpret the `.planet.jupiter` selector as `.jupiter` only. This has been fixed in IE 7.

Whereas classes are meant to reference more than one element, `ids` are meant to reference only one element in a document.

ID Selectors

`id` selectors are unique identifiers; an `id` is meant to be unique, defined once per document. Like `class` selectors discussed in the previous section, a special character precedes `id` selectors in a style sheet. To reference an `id`, you precede the `id` name with a hash mark (or pound sign, #). Like class names, this name cannot contain spaces. You should use names that only include letters, numbers, and spaces for compatibility with the older browsers. You see how this is done in Figure 3-8.

```
#jupiter {
  left: 0;
}
```

An `id` selector begins with a hash mark (or, if you prefer, the pound sign "#"), then, like the `class` selector, the hash mark is followed by the `id` name of the element.

Figure 3-8

Since there's only one Jupiter in the solar system, Jupiter lends itself as a good example of the concept of an `id` selector. Just as there is only one Jupiter in the solar system, the `id` name `jupiter` can be used only once in a document, on one element.

Browsers are forgiving of multiple `id` names per document as far as style sheets are concerned. However, using an `id` name more than once in a document can cause conflicts with other applications of unique `id` names. For example, `id` names can be used to link to a location within a document (as HTML anchors), or when referencing an element by `id` name from JavaScript. When you have an `id` name appearing more than once in the HTML document, on more than one element, the browser won't know which one you're linking to, or which one you want to refer to from JavaScript, and will have to guess. It's best to just use the `id` name for its intended purpose, just once per document.

An `id` name must be unique in so far as other `id` names are concerned. An `id` name may be repeated as a class name, should you want to do so.

The element can then be defined in the document using the `id` attribute. This is demonstrated in Figure 3-9.

```
<div id='jupiter'>
  <img src='images/jupiter.png' alt='Jupiter' class='planet' />
</div>
```

The `id` name is included in the HTML document using the `id` attribute.

Figure 3-9

You can make both `class` and `id` selectors more specific by appending the name of the element to the beginning of the selector. For instance, if in the last examples you only want `<div>` elements for each rule, the selector will look like what you see in Figure 3-10.

```
div#jupiter {
  left: 0;
}
```

Like class selectors, an `id` selector can be prepended with a type selector to make the selector apply only to a certain type of element.

Figure 3-10

Part I: The Basics

Now each rule is applied only to `<div>` elements that contain the corresponding class and id names. You may wonder why this is useful for an id selector, since an id element has to be unique in a document. Appending the selector with the type of element is useful in situations where one style sheet applies to more than one HTML document, where it's possible that you have a unique id in one of those documents that applies to, for instance, an `` element, but in another, separate document, that unique id name applies to a `<div>` element. Of course, it's best practice to avoid situations like that by making each element's id name unique, even in different documents, to avoid confusion. Sometimes, it can't be avoided. The other reason this is useful is that it makes the style sheet more intuitive and easier to follow. When you are reading a style sheet and see the id name *jupiter* but no type selector, that id can apply to any element in the document and would require you to scan the whole document from top to bottom without any other search criteria. With the type selector appended, you can narrow the search; if the element is a `<div>` element, then you know that the id selector doesn't apply to images, links, paragraphs, and so on.

Although the id must be unique, in these examples you can name only one element *jupiter*. The CSS style sheet, however, may contain as many references to that id as are necessary. The uniqueness rule only applies to naming the elements, not the references to them. You can apply classes, on the other hand, to as many elements in the body as necessary.

Now that you've had a proper introduction to the different types of things that id and class name selectors are capable of, try the following proof-of-concept exercise that lets you see how id and class selectors work.

Try It Out Class and ID Selectors

Example 3-1. To see how class and id selectors work, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Class and ID Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-1.css' />
  </head>
  <body>
    <p class='container'>
      A class represents something that you can have more than one of.
      You aptly name your class to reflect the type of item that you
      may or may not have more than one of. The class name for this
      paragraph is <i>container</i>. It could very well be that you
      have many containers, or just one.
    </p>
    <p class='container box'>
      You can chain together class names within the class attribute.
      From a purely semantic standpoint, the class names may or may
      not have a relationship with each other. Here, the class names
      are <i>container</i> and <i>box</i>. It could be said that boxes
      and containers are related, since <i>box</i> is a type of
      <i>container</i>.
```

```

    </p>
    <p class='container tank'>
        It is wise to put thought behind the naming conventions you use
        within a document. Here, <i>tank</i> is another type of
        <i>container</i>. All containers have some properties in common.
        Dimensions, color, volume, etc. But some containers may have
        properties that are unique to that container. Perhaps it has a
        different color, or capacity, or is intended to hold a different
        kind of material.
    </p>
    <p class='container' id='container-1234'>
        An id is used but once per document. Semantically speaking, the
        id should be able to identify uniquely, and be descriptive.
        You may have several containers, but only one container has the id
        <i>1234</i>. Since only one container is named <i>1234</i>, it
        becomes easier to find that container among the others.
    </p>
</body>
</html>

```

2. Save the preceding document as `Example_3-1.html`.

3. Enter the following style sheet into your text editor:

```

body {
    font-family: sans-serif;
}
p.container {
    border: 1px solid rgb(29, 179, 82);
    background: rgb(202, 222, 245);
    padding: 10px;
    width: 245px;
    height: 245px;
    float: left;
    margin: 10px;
}
p.box {
    border: 1px solid rgb(69, 199, 115);
    background: rgb(164, 201, 245);
}
p.tank {
    border: 1px solid rgb(107, 214, 145);
    background: rgb(124, 180, 245);
    clear: left;
}
p#container-1234 {
    border: 1px solid rgb(154, 232, 181);
    background: rgb(82, 157, 245);
}

```

4. Save the preceding style sheet as `Example_3-1.css`. Figure 3-11 shows what Example 3-1 looks like when rendered in Safari. You should see something similar in Firefox, IE 6, IE 7, and Opera.

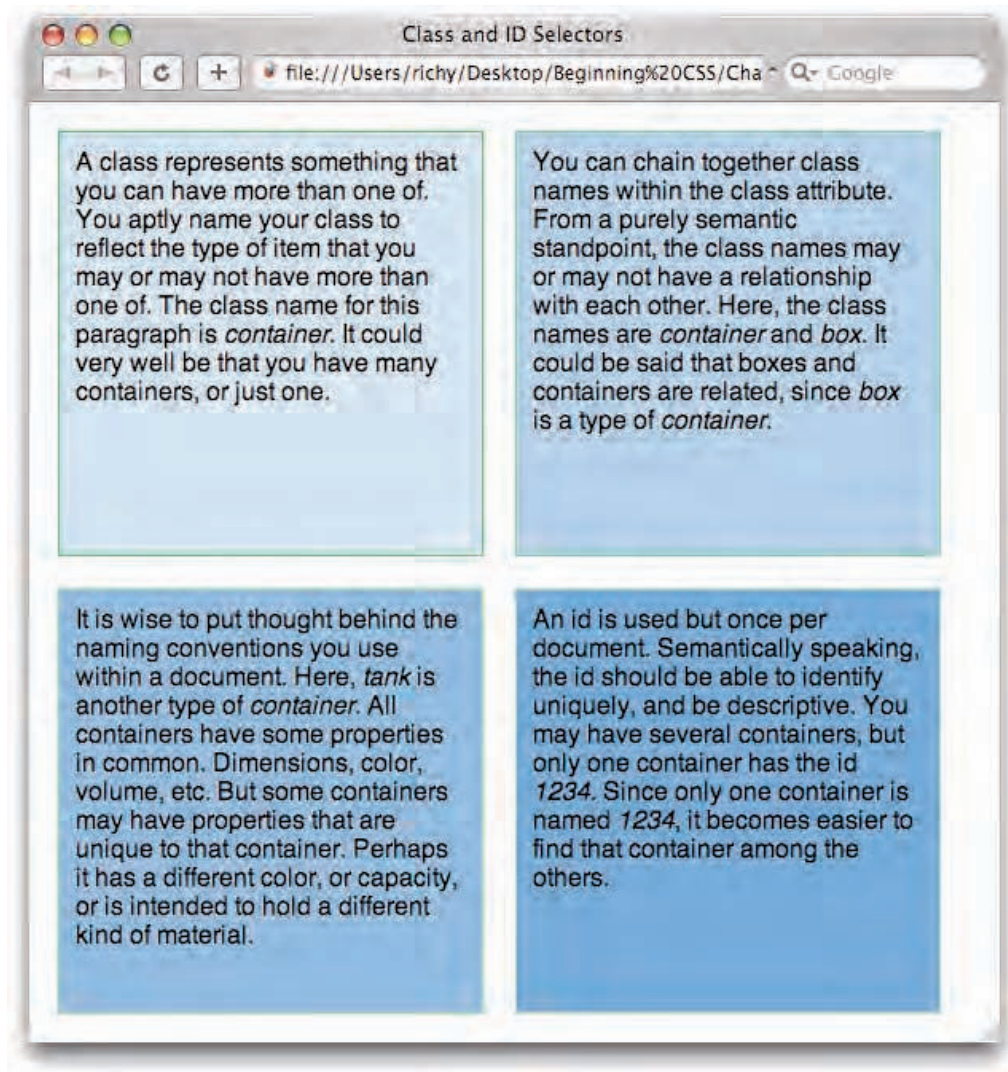


Figure 3-11

How It Works

In Example 3-1, you put your newly acquired class and id selector skills to use. The following is a rule-by-rule review of the relevant `class` and `id` styles you applied in `Example_3-1.css`.

First, you created a rule that is applied to all four `<p>` elements, since all four `<p>` elements have a class name of `container`. You were able to select all four elements because each `<p>` element in the document has a `container` class name in the value of the `class` attribute that appears on all four `<p>` elements.

```
p.container {  
  border: 1px solid rgb(29, 179, 82);  
  background: rgb(202, 222, 245);  
  padding: 10px;  
  width: 245px;
```



```
height: 245px;
float: left;
margin: 10px;
}
```

Since the preceding rule applied to all four `<p>` elements, it set common properties such as dimensions using the `width`, `height`, `padding`, `border`, and `margin` properties. You learn more about these properties in Chapter 7, “The Box Model.” For now, just examine how the `p.container` selector is working to select the elements, rather than the actual styling being applied.

In the next rule, you selected the next `<p>` element that also has two class names, `box` and `container`.

```
p.box {
  border: 1px solid rgb(69, 199, 115);
  background: rgb(164, 201, 245);
}
```

Although you could have chained the class names in the style sheet by using the selector `p.container.box`, you avoid doing this since there are known problems with this approach in IE 6. IE 6, on the other hand, supports just fine multiple class names in the `class` attribute. Referencing just the `box` class name allows you to select the element, too. You give the element a slightly richer shade of light blue, and a slightly lighter green border than was specified in the previous rule, which referenced all four `<p>` elements by the class name, `container`. You see that the `background` and `border` declarations set here overrode the previously set `background` and `border` declarations in the first container rule; you’ll learn more about this in Chapter 4, “The Cascade and Inheritance.”

In the next rule, you set properties on the `<p>` element with both the class names `container` and `tank`. Again, you gave the element an even richer light blue background (compared to the last rule, which was applied to the `<p>` element with `container` and `box` class names).

```
p.tank {
  border: 1px solid rgb(107, 214, 145);
  background: rgb(124, 180, 245);
  clear: left;
}
```

In the last rule, you used an `id` selector to select the fourth `<p>` element, which has an `id` attribute set with a value of `container-1234`. For the fourth `<p>` element, there is an even richer still light blue background, and an even lighter green border around it.

```
p#container-1234 {
  border: 1px solid rgb(154, 232, 181);
  background: rgb(82, 157, 245);
}
```

Now that you have worked through this simple, proof-of-concept demonstration of class and `id` selectors for yourself, continue to the next section, which discusses the universal, or wildcard selector.

The Universal Selector

The *universal selector* is an asterisk. When used alone, the universal selector tells the CSS interpreter to apply the CSS rule to all elements in the document. Figure 3-12 shows what a universal selector looks like.

A CSS rule is shown with an asterisk in a purple circle as the selector, followed by a curly brace containing the property and value: `border: 1px solid black;`, and a closing curly brace.

An asterisk is a wildcard selector or *universal selector*. When included alone, all elements in the document are selected.

Figure 3-12

This rule is applied to all elements contained in the document. The universal selector applies to everything, including form input fields and tables of data. It applies style to any and every element present in a document.

Try It Out The Universal Selector

Example 3-2. To see how the universal selector works, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Class and ID Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-2.css' />
  </head>
  <body>
    <h1>Universal Selectors</h1>
    <p>
      Universal selectors are wildcard selectors.
    </p>
    <p>
      When a universal selector is used alone, all elements
      within a document are selected.
    </p>
    <p>
      Even form elements are selected.
    </p>
    <form method='post' action='Example_3-2.html'>
      <fieldset>
        <legend>Feedback Form</legend>
        <table>
          <tbody>
```

```

        <tr>
            <td><label for='topic'>Topic:</label></td>
            <td><input type='text'
                name='topic'
                id='topic'
                value='Universal Selectors'
                size='25' />
            </td>
        </tr>
        <tr>
            <td><label for='feedback'>Feedback:</label></td>
            <td>
<textarea cols='55' rows='10' name='feedback' id='feedback'>
Universal selectors have some practical applications.
For instance, when debugging styles you can select
all elements and apply a border to see dimensions.
This could help you identify rogue elements causing
undue disorder in a document.
</textarea>
            </td>
        </tr>
    </tbody>
</table>
</fieldset>
</form>
</body>
</html>

```

2. Save the preceding document as `Example_3-2.html` and load it into your favorite browser.
3. Enter the following CSS into a new document in your text editor.

```

body {
    font-family: sans-serif;
}
* {
    border: 1px solid yellowgreen;
    color: green;
    padding: 5px;
    font-weight: normal;
    font-size: 12px;
}

```

4. Save the preceding styles as `Example_3-2.css`. After loading Example 3-2 into your browser, you should see output similar to that of Figure 3-13.

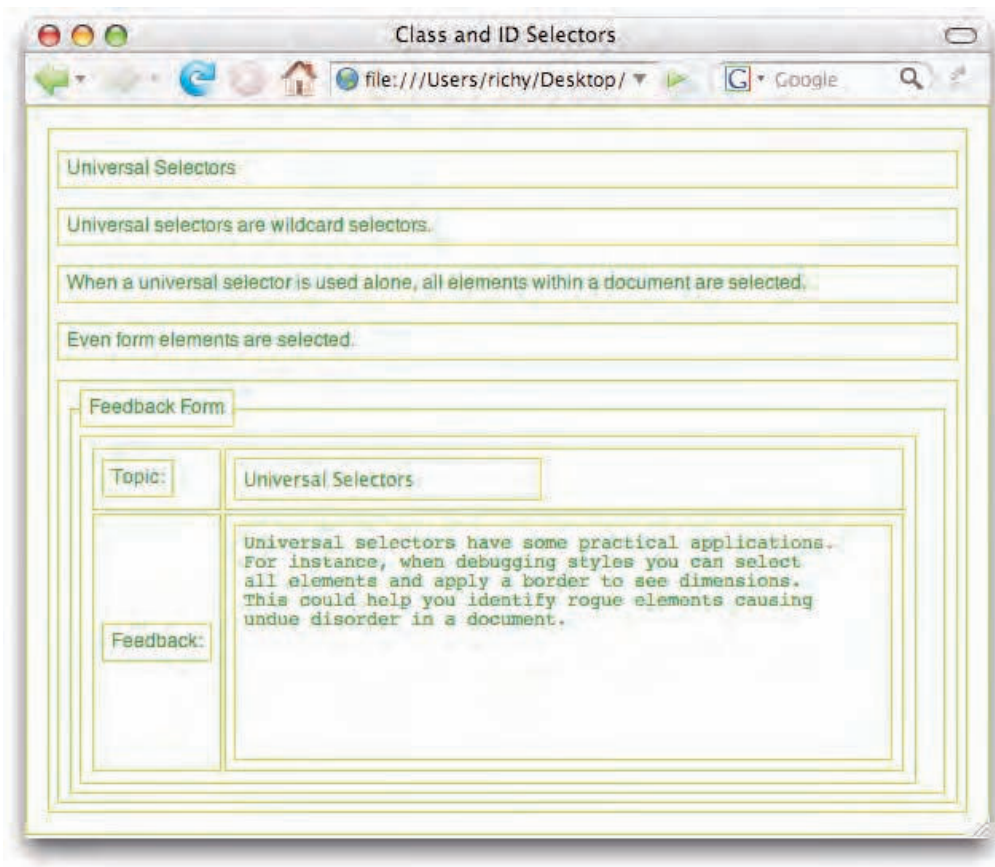


Figure 3-13

Figure 3-13 shows the results from Mac Firefox. Safari 2.0 produces similar results; the difference being only the font color is applied to the form elements. Safari 2.0 does not support custom styling of form elements very well. However, later versions have made progress in this area. IE 6 and IE 7 also differ slightly from the output here, in that the `<label>` elements are missing the top border, which is because of a bug in IE. While the results are not perfect from browser to browser, you get the idea of what the universal selector does.

How It Works

The concepts at play in Example 3-2 are very simple; the universal selector is included in the style sheet as an asterisk. The declarations in the rule that follow the asterisk are applied to all of the elements that appear in the document, provided that element is allowed to have the property in question applied. For instance, the `<tbody>` and `<tr>` elements do not accept most visual styles (borders, padding, and dimensions, for example). The universal selector, alone, doesn't have much practical application, although as previously mentioned, it can be helpful for debugging styles and highlighting element dimensions in complex documents. By applying a border to all elements, you are able to immediately see the space an element occupies.

The universal selector can also be used with other kinds of selectors, such as contextual selectors, also known as descendant selectors.

Descendant Selectors

Descendant selectors apply style based on whether one element is a descendant of another. In CSS, *descendant* means an element that is a child, grandchild, great grandchild, and so on, of another element. This type of relationship is referred to as an *ancestral* relationship. Take for example the document in Figure 3-14. If you were looking to map the ancestral relationship between the elements in Figure 3-14, you would see a tree like that in Figure 3-15.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Tree</title>
  </head>
  <body>
    <div id='heading'>
      <h1>Ancestral Relationships Among HTML Elements</h1>
      <p>
        A study of lineage in angle-bracket documents.
      </p>
    </div>
    <div id='body'>
      <p>
        Heritage is important in HTML documents. One element
        that all elements have in common is the
        <span class='inline-code'>&lt;html&gt;</span>.
      </p>
      <p>
        Ancestral relationships affect another CSS feature,
        <i>inheritance</i>, which I discuss in Chapter 4.
      </p>
      <p>
        <a href='http://p2p.wrox.com'>Wrox P2P</a> is a great
        place to go when you have technical questions.
      </p>
    </div>
  </body>
</html>
```

Figure 3-14

As a web designer, you get used to visualizing markup documents as a tree. Perhaps not as a real tree, as you see in Figure 3-15, but visualizing the lineage of an element. The concept of ancestral relationships between elements is a fundamental cornerstone to web development, and as you read on throughout this chapter and Chapter 4, you'll see that ancestral relationships play a large role in CSS development.

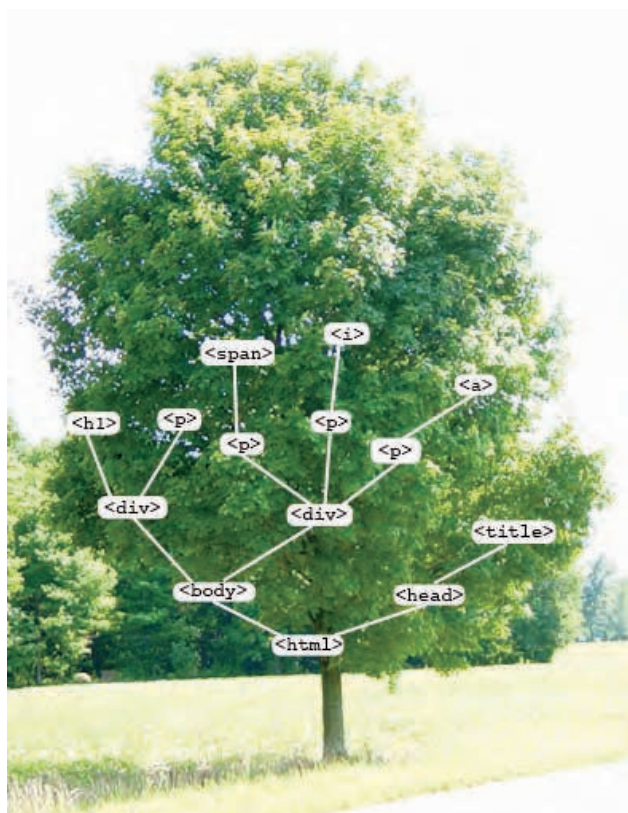


Figure 3-15

Descendant selectors apply style based on the lineage of an element. Keeping in mind the markup presented in Figure 3-14, one example of a descendant selector appears in Figure 3-16.

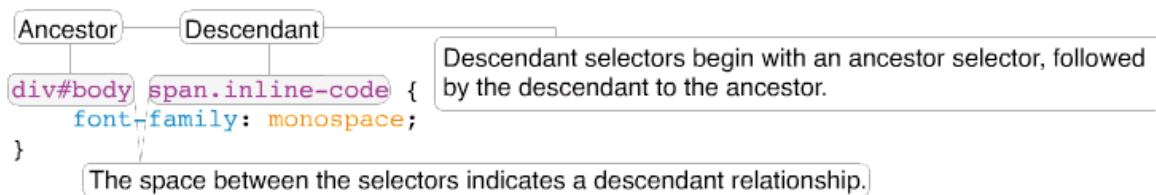


Figure 3-16

Descendant selectors are used to select an element based on the context it appears in the document. In the example code in Figure 3-16, you select a `` element with an `inline-code` class name, and apply the `monospace` font to it, but only if the `inline-code` `` element is a descendant of the `<div>` element with a `body` id name.

Descendant selectors aren't limited to just two elements; you can include more elements in the ancestral lineage, if it suits your needs. Each selector in a descendant selector chain must be separated by a space. This is demonstrated in code in Figure 3-17.

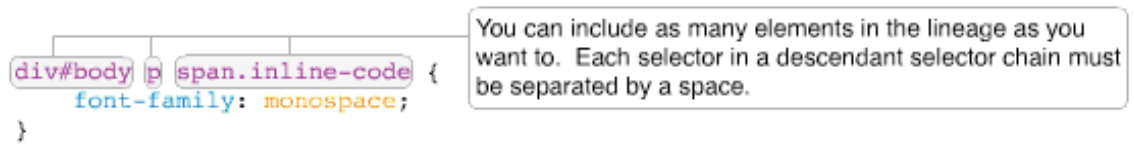


Figure 3-17

In fact, the entire lineage from the eldest ancestor, the `<html>` element, down through the generations to the element you want to select, can be included in a descendant selector chain.

Descendant selectors can also be combined with the universal selector. You can see an example of this in Figure 3-18.

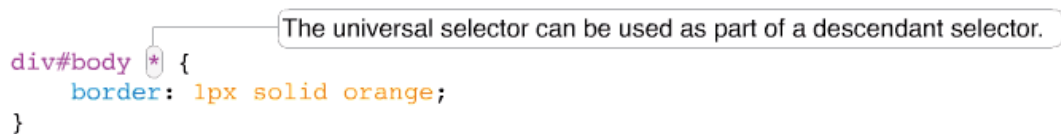


Figure 3-18

The universal selector can appear in any part of a descendant selector. When it is included, it is a wildcard. In Figure 3-18, you select all descendants of the `body <div>` element.

Because descendant selectors are part of the oldest CSS 1 specification, they are the widest supported contextual selector. The upcoming sections (through to the section titled “Attribute Selectors”) are CSS 2 selectors, which are not supported by IE 6.

In the CSS level 1 specification, descendant selectors are referred to as contextual selectors. The name change was made in the CSS level 2 specification. The name change likely resulted from new selectors in CSS 2, several of which can also be considered contextual because their selection is based on the context in which the target element appears in the document.

Try It Out Descendant Selectors

Example 3-3. To see how descendant selectors work, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Descendant Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-3.css' />
  </head>
  <body>
    <h1>Descendant Selectors</h1>
    <p>
      Descendant selectors apply styles based on ancestral relationships.
```

```
The first descendant example I present applies style to the
<span class='code'>&lt;span&gt;</span> element named <em>code</em>,
which is a descendant of <span class='code'>&lt;p&gt;</span> elements.
To do this, the selector <span class='code'>p span.code</span> is used.
</p>
<p>
    Using CSS, styles can be applied to any number of documents. Since
    this is the case, there may be <span class='code'>&lt;span&gt;</span>
    elements with a class name of <em>code</em> in several documents, but
    have different styles applied depending on the context it appears,
    which is the exact situation the inventors of the descendant
    selector had in mind when it was conceived.
</p>
<p class='note'>
    The note text is given different styles. To do this another descendant
    selector is used. This time the selector is <span class='code'>p.note
    span.code</span>
</p>
</body>
</html>
```

2. Save the preceding document as `Example_3-3.html`.
3. Enter the following CSS in a new document in your text editor:

```
body {
    font-face: sans-serif;
}
h1 {
    margin: 5px;
}
p {
    border: 1px solid rgb(200, 200, 200);
    background: rgb(234, 234, 234);
    padding: 5px;
    margin: 5px;
}
p.note {
    background: yellow;
    border: 1px solid gold;
}
span.code {
    font-family: monospace;
    padding: 0 10px;
}
p span.code {
    background: yellow;
}
p.note span.code {
    background: lightyellow;
}
```

4. Save the preceding CSS as `Example_3-3.css`. This example results in the output you see in Figure 3-19.

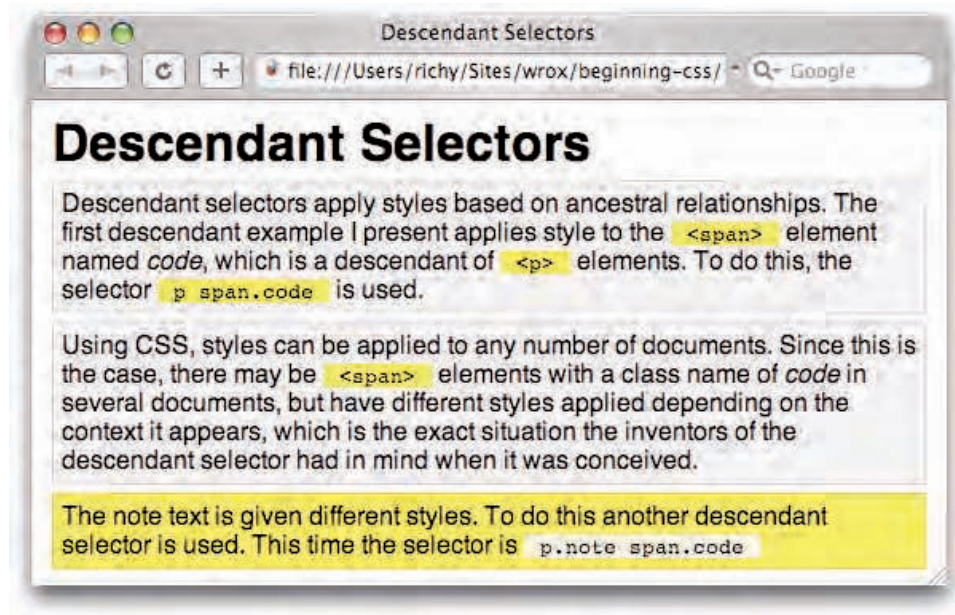


Figure 3-19

How It Works

Descendant selectors apply style based on an ancestral relationship. The first example of descendant selectors that you see in Example 3-3 is `p span.code`. This selector selects `` elements with class names of *code*, but only when they appear as descendants of `<p>` elements. That is to say, when a `` element exists in the document and it has a class name of *code*, and it is the child, grandchild, great grandchild, and so on, of a `<p>` element, those elements receive a yellow background.

The second example of descendant selectors in Example 3-3 is `p.note span.code`, where two type and class selectors are included in a descendant selector. In this selector any `<p>` elements appearing in the document with a class name of *note* that have descendant `` elements, which have a class name of *code*, receive lightyellow backgrounds.

Descendant selectors allow you to apply style based on ancestral relationships. In the next section, you see a similar selector, the direct child selector, which also applies style based on an ancestral relationship, but a narrower, more specific ancestral relationship, parent and child.

Direct Child Selectors

Direct child selectors operate much like descendant selectors in that they also rely on an ancestral relationship to decide where to apply style. Descendant selectors, however, are more ambiguous because they apply to any descendant of an element; the descendant can be a grandchild or a great-grandchild, or a great-great-grandchild, and so on. Direct child selectors apply only to immediate children of the element. This is achieved by introducing a new syntax for the selector. Figure 3-20 is an example of a direct child selector.

Part I: The Basics

IE 6 does not support direct child selectors natively; see this book's website at www.wrox.com/go/beginning_css2e for compatibility help.

```
p > span.inline-code {  
    font-family: monospace;  
}
```

Like descendant selectors, direct child selectors are chained together, but instead of a space, a greater-than symbol (or right angle bracket) is used to separate each element in the selector.

Figure 3-20

In Figure 3-20 you see that the greater than sign (or right angle bracket), `>`, is used in the style sheet to select an element in the HTML document. In Figure 3-20, you see a parent/child relationship in the direct child selector, `p > span.inline-code`. In order to apply the declaration `font-family: monospace;`, the `` element with the class name `inline-code`, must be the child of a `<p>` element.

Direct child selectors are selectors that depend on the context that an element appears in a document. The context in this case is a parent/child relationship. Like descendant selectors, a direct child selector chain can have as many elements as you like; an example of this is shown in Figure 3-21.

```
div#body > p > span.inline-code {  
    font-family: monospace;  
}
```

You can include as many elements in a direct child selector chain as necessary.

Figure 3-21

In Figure 3-21, you see two parent/child relationships represented in one selector. The `<p>` element is a direct child of the `<div>` with an id name of `body` and the `` with a class name of `inline-code` is a direct child of the `<p>` element.

You can also mix selectors, if you have need of it. Figure 3-22 shows mixing descendant selectors with direct child selectors.

```
div#body > p span.inline-code {  
    font-family: monospace;  
}
```

Direct child and descendant selectors can be mixed together.

Figure 3-22

In fact, you can mix and match selectors in pretty much any way imaginable—direct child selectors with descendant selectors, with universal selectors. CSS is very flexible in this regard, provided browser support for the selector exists.

Try It Out Direct Child Selector

Example 3-4. To see how the direct child selectors work, follow these steps.

1. Using the markup in `Example_3-3.html`, make the following highlighted changes:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Direct Child Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-4.css' />
  </head>
  <body>
    <h1><ins>Direct Child</ins> <del>Descendant</del> Selectors</h1>
    <p>
      <ins>Direct Child</ins> <del>Descendant</del> selectors apply styles
      based on <ins>parent/child</ins> <del>ancestral</del> relationships.
      The first <ins>direct child</ins> <del>descendant</del> example I
      present applies style to the
      <span class='code'>&lt;span&gt;</span> element named <em>code</em>,
      which is a <del>descendant</del> <ins>child</ins> of
      <span class='code'>&lt;p&gt;</span> elements.
      To do this, the selector <span class='code'>p <ins>&gt;</ins>
      span.code</span> is used.
    </p>
    <p>
      Using CSS, styles can be applied to any number of documents. Since
      this is the case, there may be <span class='code'>&lt;span&gt;</span>
      elements with a class name of <em>code</em> in several documents, but
      have different styles applied depending on the context it appears,
      which is the exact situation the inventors of the <del>descendant</del>
      <ins>child</ins> selector had in mind when it was conceived.
    </p>
    <p class='note'>
      The note text is given different styles. To do this another
      <del>descendant</del> <ins>direct child</ins>
      selector is used, this time the selector is
      <span class='code'>p.note <ins>&gt;</ins> span.code</span>
    </p>
  </body>
</html>
```

2. Save the preceding markup document as `Example_3-4.html`.
3. Using the style sheet that you made for Example 3-3, `Example_3-3.css`, make the following highlighted changes.

```
body {
  font-face: sans-serif;
}
h1 {
  margin: 5px;
}
```

```
del {
    color: crimson;
}
ins {
    color: forestgreen;
}
p {
    border: 1px solid rgb(200, 200, 200);
    background: rgb(234, 234, 234);
    padding: 5px;
    margin: 5px;
}
p.note {
    background: yellow;
    border: 1px solid gold;
}
span.code {
    font-family: monospace;
    padding: 0 10px;
}
p > span.code {
    background: yellow;
}
p.note > span.code {
    background: lightyellow;
}
```

4. Save the preceding style sheet as `Example_3-4.css`. The preceding example results in the rendered document pictured in Figure 3-23.

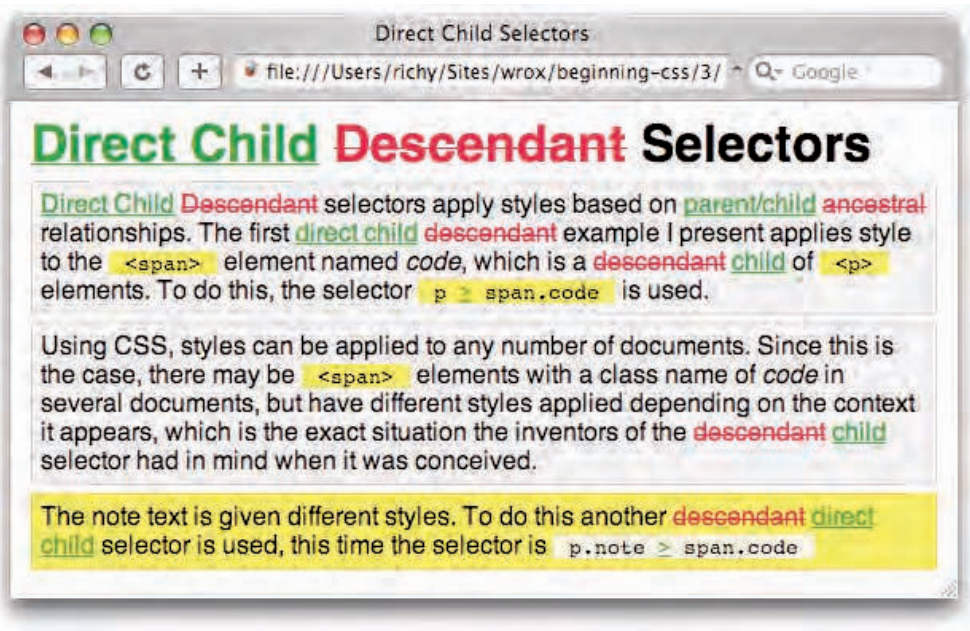


Figure 3-23

How It Works

As is illustrated in Example 3-4, the direct child selector is pretty similar to the descendant selector. In most situations you can get away with using a descendant selector where a child selector could be used and vice versa, the only difference being the direct child must be a parent/child relationship, and the descendant selector can be a more ambiguous ancestral relationship. Using a descendant selector, you have greater compatibility since IE 6 does not support the direct child selector (at least, not without a workaround, which you can find in Chapter 16, available at www.wrox.com/go/beginning_css2e).

There are some situations where a descendant selector would not be desired, and a direct child selector would come in handy, or it wouldn't exist. Those situations are a bit too complex to explain properly here, in addition to being rare.

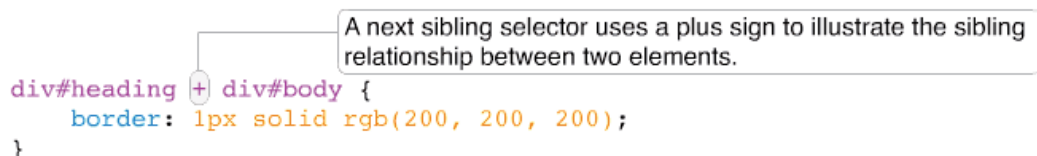
In Example 3-4, you see that the direct child selector uses a greater than sign (>) within the selector to indicate the parent/child relationship, whereas the descendant selector you saw in Example 3-3 uses space between selectors to indicate an ancestral relationship, which is not limited to parent/child, but could indicate grandparent/grandchild, great-grandparent/great-grandchild, and so on.

Selecting a child element based on the element's parent can be helpful. These contextual selectors allow developers to define fewer class and id names in a markup document, and instead select elements based on the context they appear in a document. In the next section I present another contextual selector, the direct adjacent sibling combinator (its official name), or *next sibling* for short (because that's just too long!).

Next Sibling Selector

The official name of the selector I discuss in this section, according to the W3C is the *adjacent sibling combinator*. I think that's too long and complicated, so I've shortened it to just *next sibling*. The next sibling selector selects, surprise, an element's next sibling. Looking back on the markup in Figure 3-14, the markup in Figure 3-24 is a demonstration of what a next sibling selector looks like in a style sheet.

IE 6 does not support next sibling selectors natively; see this book's website at www.wrox.com/go/beginning_css2e for compatibility help.



```
div#heading + div#body {
  border: 1px solid rgb(200, 200, 200);
}
```

Figure 3-24

In Figure 3-24, you see that a plus sign is used to denote the sibling relationship between two elements. You may be thinking to yourself at this point, *well that's just fine and dandy, but what's the practical application? Can't you just reference the div#body alone and get the same result? Why do you need a next sibling selector?* I'm glad you asked. This selector can be useful in certain situations, such as when you have several HTML documents that reference the same style sheet. In some of these documents, the <div> with an id

Part I: The Basics

name of heading and the <div> with an id name of body are siblings, and they appear in the source one right after the other. In other documents these two elements may not be siblings. Naturally, if you have different template requirements in these theoretical two different kinds of documents, you may like to have a way to reference the ones where these elements are siblings explicitly, and that is one example of a practical application of the next sibling selector. Also, as I mentioned in the previous section, “Direct Child Selectors,” sometimes you want to avoid creating new id and class names. In some situations when you use the next sibling selector, you can potentially avoid creating new class and id names.

In the following proof-of-concept example, you try out the next sibling selector for yourself.

Try It Out Next Sibling Selector

Example 3-5. To see how the next sibling selector works, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Next Sibling Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-5.css' />
  </head>
  <body>
    <h1>Next Sibling Selectors</h1>
    <p>
      The next sibling selector (or adjacent sibling combinator as
      it's officially called) allows you to select an element based on
      its sibling. This paragraph has a lightyellow background and
      darkkhaki text.
    </p>
    <p>
      This paragraph has a yellowgreen background and green text.
    </p>
    <p>
      This paragraph has no colored background, border, or text.
    </p>
  </body>
</html>
```

2. Save the preceding markup as Example_3-5.html.
3. Enter the following CSS into your text editor:

```
body {
  font: 12px sans-serif;
}
p {
  padding: 5px;
}
h1 + p {
  background: lightyellow;
  color: darkkhaki;
  border: 1px solid darkkhaki;
```

```

}
h1 + p + p {
    background: yellowgreen;
    color: green;
    border: 1px solid green;
}

```

4. Save the style sheet as `Example_3-5.css`. Once loaded into your next sibling selector supporting browser, you should see something like that in Figure 3-25.

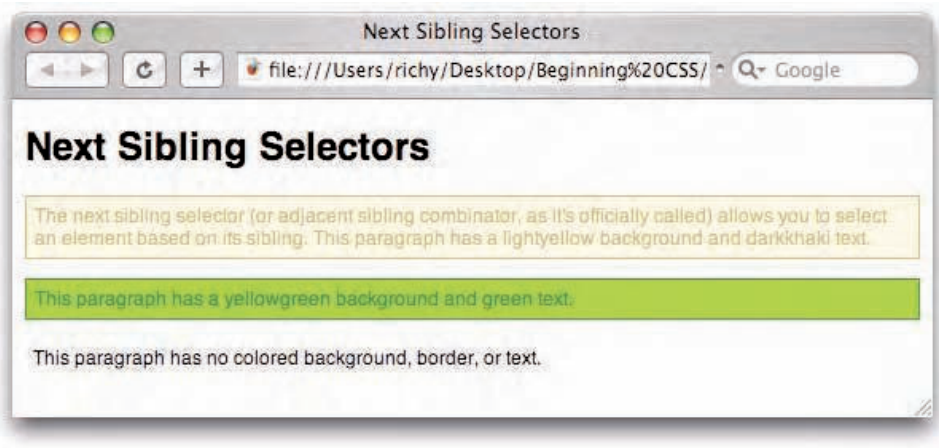


Figure 3-25

How It Works

The next sibling selector applies a style based on a sibling relationship. The following is a review of the relevant styles that you applied in `Example_3-5.css`.

The first style you applied in `Example_3-5.css` is applied to the first paragraph in `Example_3-5.html`. The selector `h1 + p` means that if a `<p>` element is the next, directly adjacent sibling to an `<h1>` element, apply the declarations in this rule.

```

h1 + p {
    background: lightyellow;
    color: darkkhaki;
    border: 1px solid darkkhaki;
}

```

The rule only applies when a `<p>` element is the directly adjacent sibling of an `<h1>` element.

In the second rule, you have a more complex next sibling selector. It says that if a `<p>` element is the directly adjacent sibling of another `<p>` element, which in turn is the directly adjacent sibling to an `<h1>` element, apply the declarations in the rule.

```

h1 + p + p {
    background: yellowgreen;
}

```

```
color: green;
border: 1px solid green;
}
```

Just as the direct child selector allows you to apply a style based on a parent/child relationship, next sibling selectors allow you to apply style based on a sibling relationship.

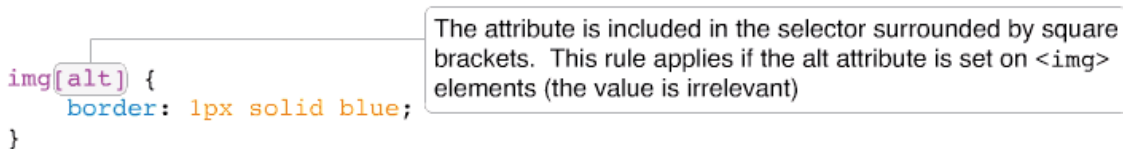
Sometimes, it's useful to have a selector that can apply styles based on the existence or value of an element's attributes.

Attribute Selectors

Attribute selectors are used to apply style sheet declarations based on the presence of attributes or attribute values of an HTML element.

IE 6 does not support attribute selectors natively; see this book's website at www.wrox.com/go/beginning_css2e for compatibility help.

Figure 3-26 is an example of an attribute selector that applies a style sheet rule based on the presence of an attribute.



```
img[alt] {
  border: 1px solid blue;
}
```

The attribute is included in the selector surrounded by square brackets. This rule applies if the alt attribute is set on elements (the value is irrelevant)

Figure 3-26

In Figure 3-26, if the alt attribute is set on elements, those elements receive a blue border. Detecting the presence of an alt attribute is good practice, since the alt attribute is required on all elements per the HTML 4.01 specification. When the rule in Figure 3-26 is used, elements that don't have a blue border need an alt attribute applied.

You are not limited to detecting the presence of an attribute; there are several types of attribute selectors, and CSS is capable of detecting attributes based on the following criteria:

- ☐ The presence of an attribute
- ☐ The value of an attribute
- ☐ Whether one of several possible values is present in an attribute
- ☐ Whether the attribute value begins with a specific string
- ☐ Whether the attribute value ends with a specific string
- ☐ Whether the attribute value contains a specific string anywhere in the value, be it at the beginning, end, or middle

The following sections examine each type of attribute selector in greater depth and provide examples of the syntax for each.

Selection Based on the Value of an Attribute

Attribute value selectors delegate style declarations based on an attribute's presence and value. In Figure 3-27, you see an example of what the syntax looks like to select an element based on an attribute's presence and value.

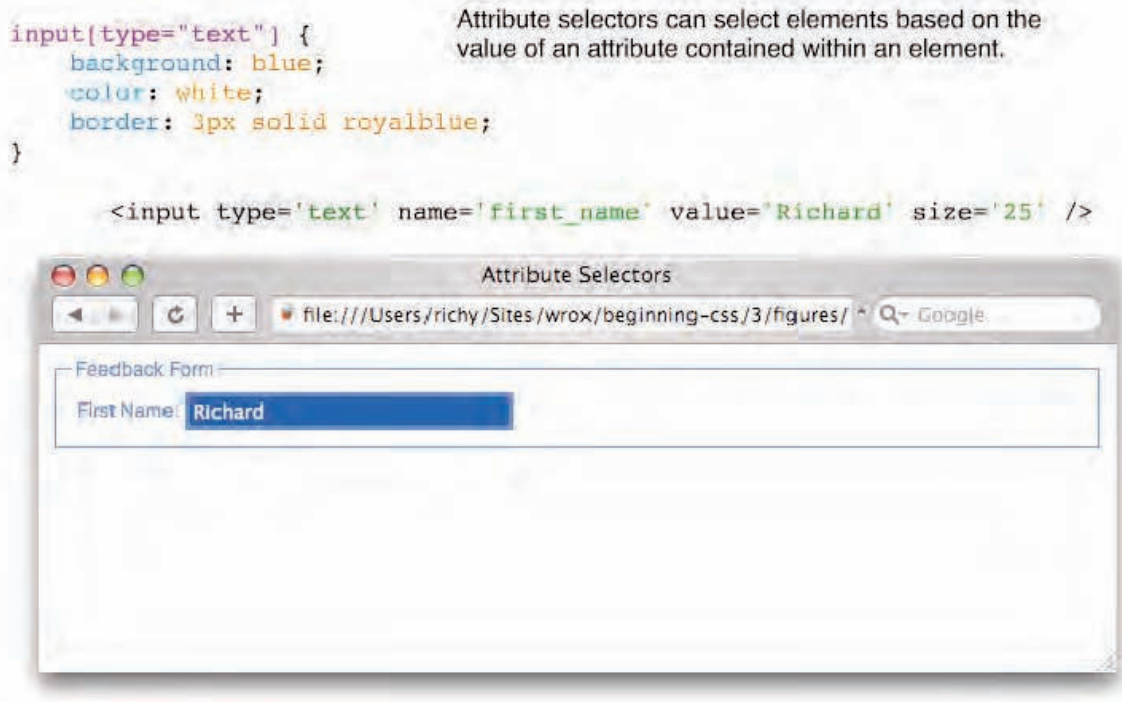


Figure 3-27

In Figure 3-27, you see how to select a text `<input>` element based on the presence of the attribute `type` and a value of `text`.

You are not limited to the presence of only one attribute. An element may also be selected based on the presence and value of multiple attributes, which you see an example of in Figure 3-28.

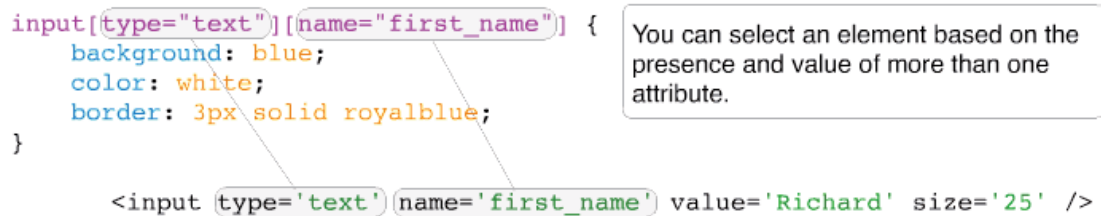


Figure 3-28

Part I: The Basics

In Figure 3-28, you see a rule that selects an element based on the presence and value of two attributes: the `type` and `name` attributes. In Figure 3-28, when the `type` attribute is `text` and the `name` attribute is `first_name`, the declarations in the rule are applied to that element. Attribute selectors let you avoid the need of setting class or id selectors when they are otherwise unnecessary.

In the following example, you try out attribute value selectors for yourself.

Try It Out Attribute Value Selectors

Example 3-6. To see how attribute value selectors work, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Attribute Selectors</title>
    <link rel='stylesheet' type='text/css' href='Figure_3-28.css' />
  </head>
  <body>
    <form method='post' action='Example_3-3.html'>
      <fieldset>
        <legend>Feedback Form</legend>
        <table>
          <tbody>
            <tr>
              <td>
                <label for='first-name'>First Name:</label>
              </td>
              <td>
                <input type='text'
                  name='first_name'
                  id='first-name'
                  value='Richard'
                  size='25' />
              </td>
            </tr>
            <tr>
              <td>
                <label for='last-name'>Last Name:</label>
              </td>
              <td>
                <input type='text'
                  name='last_name'
                  id='last-name'
                  value='York'
                  size='25' />
              </td>
            </tr>
            <tr>
              <td>
                <label for='account-password'>Password:</label>
              </td>
              <td>
```



```

                                <input type='password'
                                    name='password'
                                    id='account-password'
                                    size='25'
                                    value='mypass' />
                                </td>
                            </tr>
                        </tbody>
                    </table>
                </fieldset>
            </form>
        </body>
    </html>

```

2. Save the markup as `Example_3-6.html`.
3. Enter the following CSS into a new document in your text editor:

```

* {
    font: 12px sans-serif;
    padding: 5px;
    color: royalblue;
}
fieldset {
    border: 3px solid rgb(234, 234, 234);
    background: rgb(244, 244, 244);
}
label {
    display: block;
    text-align: right;
    width: 100px;
}
label, legend {
    background: gold;
    border: 1px solid rgb(75, 75, 75);
    color: rgb(75, 75, 75);
}
input[type='text'] {
    background: blue;
    color: lightblue;
    border: 3px solid lightblue;
}
input[type='text'][name='last_name'] {
    background: forestgreen;
    color: yellowgreen;
    border: 3px solid yellowgreen;
}
input[type='password'][name='password'] {
    background: crimson;
    color: pink;
    border: 3px solid pink;
}

```

4. Save the CSS as `Example_3-6.css`. Figure 3-29 shows what Example 3-6 looks like rendered in a browser that supports attribute selection based on value.

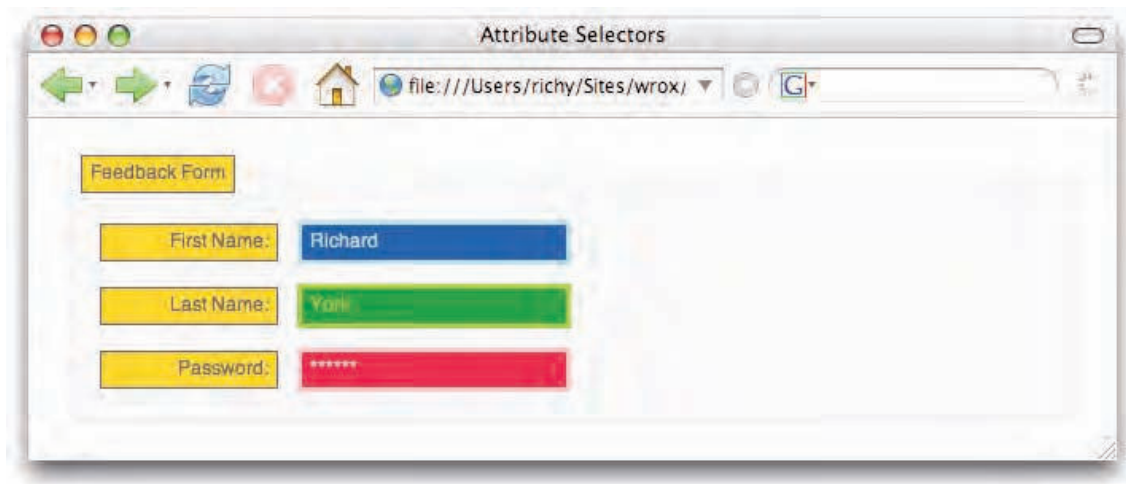


Figure 3-29

How It Works

In Example 3-6, you saw an example of the attribute selector. This type of attribute selector makes a selection based on the value of an attribute in the HTML document. Following is a review of the relevant rules in Example 3-6.

The first selector applies to all `<input>` elements that have a `type="text"` attribute. (Keep in mind that the quoting style can be either single or double quotes in either place; it doesn't matter which. Use what makes sense to you.)

```
input[type='text'] {  
    background: blue;  
    color: lightblue;  
    border: 3px solid lightblue;  
}
```

Two elements in the document match the criteria: the *First Name* and the *Last Name* `<input>` fields of the form. The preceding rule is applied only to the First Name field though, since the last name field has a rule of its own that overrides the preceding rule. The concept of overriding one rule with another is called the cascade, and you learn more about the cascade in Chapter 4. So, the preceding rule applies to this markup:

```
<input type='text' name='first_name' id='first-name' value='Richard' value='25' />
```

The preceding markup appears all on one line, whereas in the original `Example_3-6.html`, it was spread out over several lines to accommodate the width constraints of this printed text.

The preceding `<input>` field receives a blue background, the text within is colored `lightblue` via the `color` property, and a border, three pixels wide, solid and also `lightblue` goes around it.

The next rule applies to the Last Name field; it receives a `forestgreen` background, `yellowgreen` text, and a border three pixels wide, solid, and also `yellowgreen`.

```
input[type='text'][name='last_name'] {
  background: forestgreen;
  color: yellowgreen;
  border: 3px solid yellowgreen;
}
```

In the preceding rule you select the `<input>` element based on the value of *two* attributes: the `type` attribute and the `name` attribute.

Finally, in the last rule you select the `<input>` element with a `type="password"` attribute, and like the last rule, you select the element based on the value of two attributes, the `type` and `name` attributes.

```
input[type='password'][name='password'] {
  background: crimson;
  color: pink;
  border: 3px solid pink;
}
```

While selecting an attribute based on a value is useful, you can also select an attribute based on just part of the value. These are called attribute substring selectors.

Attribute Substring Selectors

Taking the flexibility of attribute selectors even further, the selectors in the following sections choose elements based on whether a particular string appears at the beginning of an attribute's value, at the end of an attribute's value, or anywhere inside an attribute's value. A string that appears inside another string is referred to as a *substring*. You can select an element based on what appears at the beginning of an attribute's value.

Selection Based on Attribute Values That Begin with a String

The first type of substring attribute selector chooses elements with an attribute value that begins with a particular string. You see an example of this in Figure 3-30.

In Figure 3-30, the rule selects `<a>` elements that have an `href` attribute. When the value of the `href` attribute begins with `ftp://`, the rule selects all of the FTP links in a web page and gives them a floppy disk icon in the background, 20 pixels of left padding so that the text of the link doesn't overlap the floppy disk icon image, and colors them `crimson`.

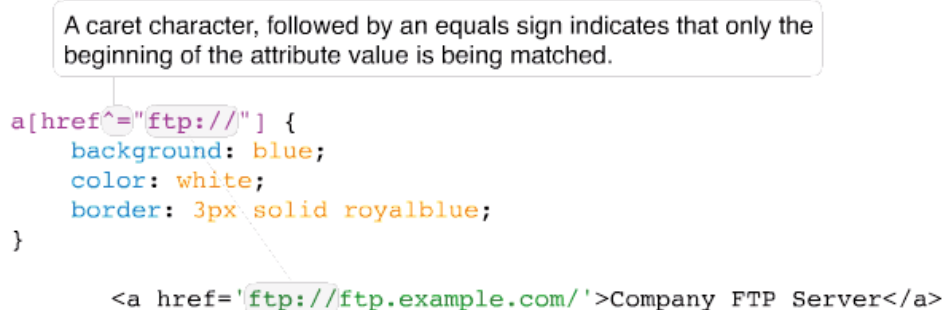


Figure 3-30a

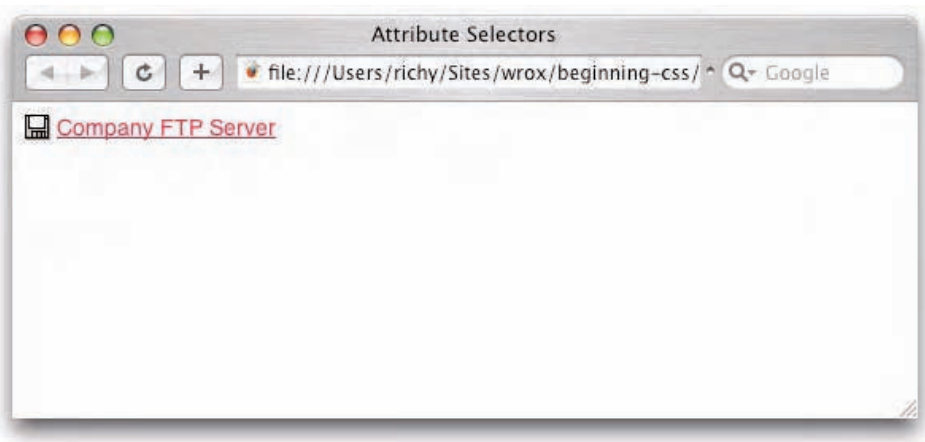


Figure 3-30b

This attribute substring selector introduces the caret (^) character in the selector syntax, which indicates that the attribute value begins with *ftp://*. Each *href* attribute prefixed with *ftp://* is then styled according to the declarations defined in the rule.

Another example of this syntax in action is to match all e-mail links in a page, and you can see an example of this in Figure 3-31.

```
a[href^="mailto:"] {  
  background: url('envelope.png') no-repeat left center;  
  padding-left: 25px;  
  color: royalblue;  
  text-decoration: none;  
}  
  
<a href="mailto:webmaster@example.com">Email the webmaster!</a>
```

Figure 3-31a



Figure 3-31b

When the selector is `a[href^="mailto:"]` you match all e-mail links within a document.

Just as you can match values that appear at the beginning of a string, you can also match values that appear at the end of a string.

Selection Based on Attribute Values That End with a String

The next substring attribute selector chooses elements with attributes whose value ends with a string. An example of this appears in Figure 3-32.

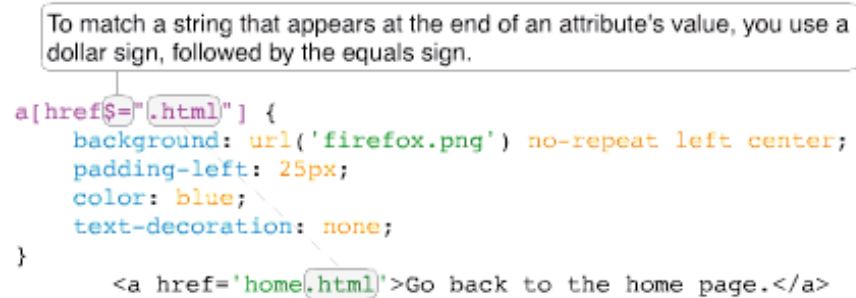


Figure 3-32a



Figure 3-32b

The selector of the preceding rule uses the dollar sign to signify that the selector matches the end of the attribute value. This changes all links that end in an `.html` suffix to blue, with a Firefox document icon, 25 pixels of left padding, and no underline.

The `href` attribute's value ends with the string `.html`, so it receives a text color of blue. Conversely, this principle does not apply to the `href` attribute of the following `<a>` element:

```
<a href='http://www.example.com/index.php'>A PHP Page</a>
```

Part I: The Basics

The attribute's value in this example ends with a .php suffix, so it does not receive a text color of blue, and a Firefox document icon, 25 pixels of left padding, and the underline are removed.

You've seen how to select an attribute's value based on what appears at the beginning and at the end of the attribute's value. The next section describes how to select an attribute's value based on the value being anywhere: at the beginning, the end, or anywhere in between.

Selection Based on Attribute Values That Contain a String

The final type of attribute substring selector is a wildcard attribute substring selector. It selects an element that contains an attribute whose value contains a string anywhere in the value: at the beginning, the end, or anywhere in the middle. This attribute substring selector uses an asterisk in the syntax to indicate that the selector is looking anywhere inside the value, as shown in Figure 3-33.

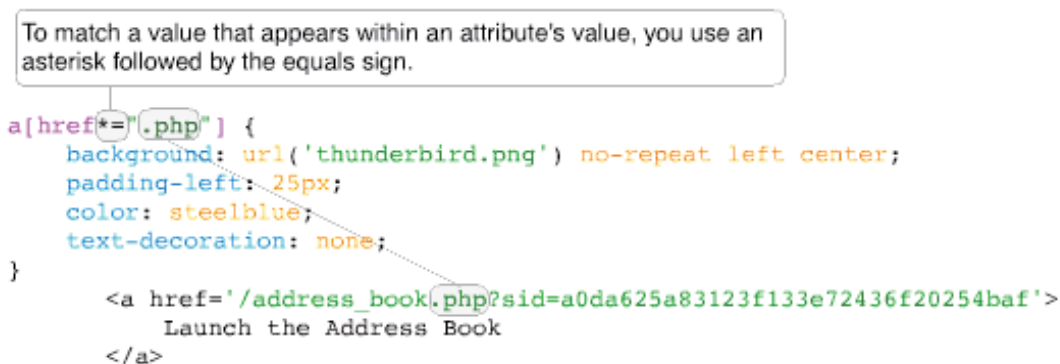


Figure 3-33a

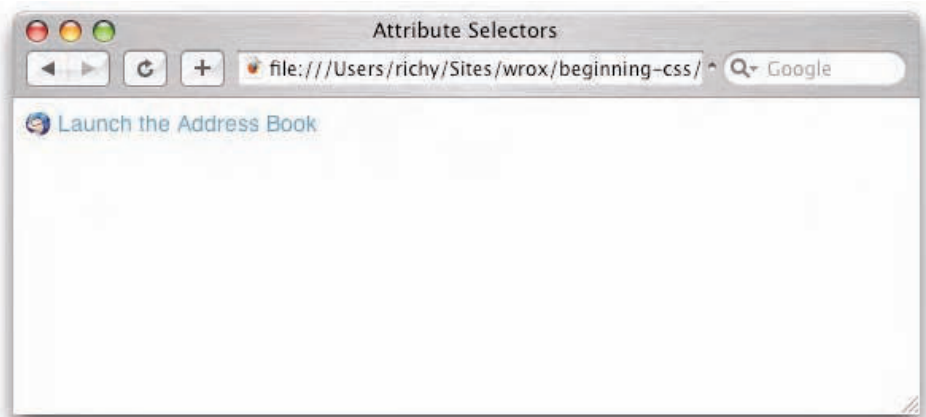


Figure 3-33b

This matches any URL that contains a .php extension regardless of whether the URL contains anchors or query strings.

All that after the question mark is called the *query string*, which holds special meaning for programming languages such as PHP, ASP, Perl, and others. What that does isn't important. What is important is that using this style sheet rule, the selector finds the .php extension even though it is in the middle of the value. The selector also finds the .php value if it appears at the beginning or the end of the URL:

```
<a href='http://www.example.com/index.php'>A .php page</a>
```

The markup presented in Figure 3-33 and in the preceding example both receive a Thunderbird icon, 25 pixels of left padding, steelblue text, and the underline removed.

In the following example, you experiment with attribute substring selectors.

Try It Out Attribute Substring Selectors

Example 3-7. To see how attribute substring selectors work, follow these steps.

1. Enter the following markup:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Attribute Substring Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-7.css' />
  </head>
  <body>
    <h1>Proof-of-Concept: Attribute Substring Selectors</h1>
    <ul>
      <li><a href='index.html'>HTML Page Link</a></li>
      <li><a href='document.pdf'>PDF Link</a></li>
      <li><a href='ftp://www.example.com/'>FTP Link</a></li>
      <li><a href='http://www.example.com/#note'>Anchor Link</a></li>
    </ul>
  </body>
</html>
```

2. Save the preceding markup as Example_3-7.html.
3. Enter the following style sheet:

```
body {
  font: 14px sans-serif;
}
h1 {
  font-size: 16px;
}
ul {
  list-style: none;
}
li {
  margin: 5px 0;
}
a {
  padding-left: 20px;
```

```
}
a[href^="ftp://"] {
    color: goldenrod;
    background: url('save.png') no-repeat left center;
}
a[href*="#"] {
    color: cadetblue;
    background: url('anchor.png') no-repeat left center;
}
a[href$=".html"] {
    color: dodgerblue;
    background: url('firefox.png') no-repeat left center;
}
a[href$=".pdf"] {
    color: red;
    background: url('pdf.png') no-repeat left center;
}
```

4. Save the preceding style sheet as `Example_3-7.css`. The preceding markup and style sheet result in the rendered output that you see in Figure 3-34.

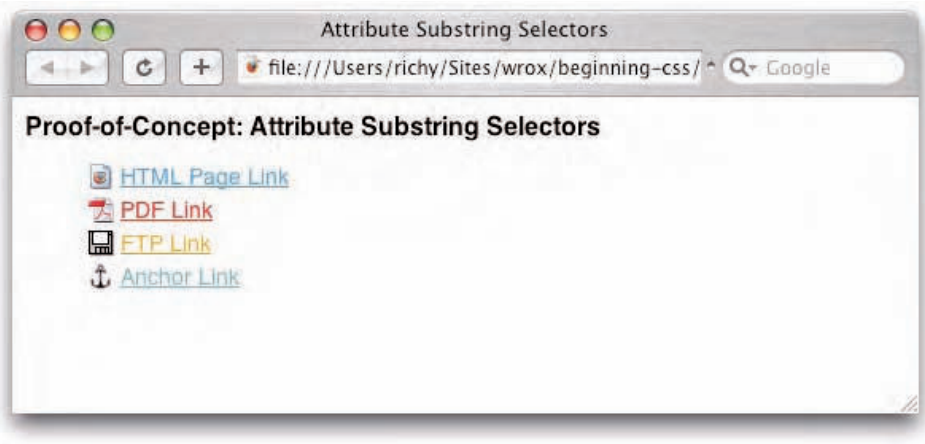


Figure 3-34

How It Works

In Example 3-7, you see how to select an attribute based on just a small portion of its value. The following is a review of the relevant attribute substring selectors.

The first attribute substring rule that you applied styles the FTP link. The selector `a[href^="ftp://"]` applies the style because the href attribute value in the HTML begins with the characters `ftp://`. To select only the beginning of the string, you used a caret character followed by the equals sign.

```
a[href^="ftp://"] {
    color: goldenrod;
    background: url('save.png') no-repeat left center;
}
```

The second attribute substring selector that you applied styles the anchor link. The selector `a[href*="#"]` finds the special hash mark (or pound sign) within the value of the `href` attribute in the HTML, `http://www.example.com/#note`. The hash character can appear anywhere in that value, and the rule still would apply `cadetblue` colored text and the `anchor.png` image to the background.

```
a[href*="#"] {  
    color: cadetblue;  
    background: url('anchor.png') no-repeat left center;  
}
```

The third attribute substring selector that you applied styles the plain old HTML document link. Because the value of the `href` attribute ends in `.html`, the color `dodgerblue` is applied as the text color, and the `firefox.png` image is applied to the background.

```
a[href$=".html"] {  
    color: dodgerblue;  
    background: url('firefox.png') no-repeat left center;  
}
```

The last attribute substring rule that you applied was just like the last, only now you are styling links to PDF documents. When the value of the `href` attribute ends in `.pdf`, the link is colored `red`, and given a PDF icon as the background image.

```
a[href$=".pdf"] {  
    color: red;  
    background: url('pdf.png') no-repeat left center;  
}
```

In the next section you begin to explore a different type of selector, pseudo-element selectors.

Pseudo-Elements `:first-letter` and `:first-line`

Pseudo-elements represent certain aspects of a document not easily modifiable with plain markup. Pseudo-elements may be used to modify the formatting of the first letter of a paragraph, or the first line of a paragraph, for example.

The pseudo-elements `:first-letter` and `:first-line` refer to the first letter and first line of an element containing text. When you design a website, it is helpful to have control over how you present content. With the `:first-letter` and `:first-line` pseudo-elements, you can control the formatting of the first letter and first line of a paragraph completely from CSS. You may add an increased font size or other font effects, apply a background color or image, or use just about any text effect supported by CSS and the browser.

You can apply pseudo-elements to a specific element, via a selector, or to all elements. Figure 3-35 shows an example of styling the first letter of a paragraph using the `:first-letter` pseudo-element.

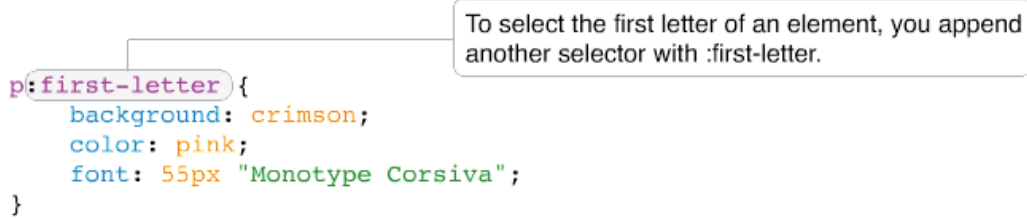


Figure 3-35a

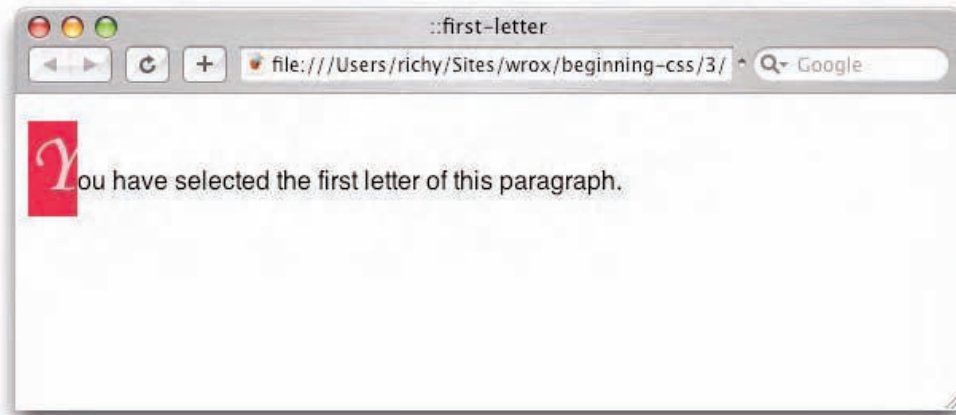


Figure 3-35b

In Figure 3-35, you see that to select the first letter in the paragraph, "Y", you use a `::first-letter` pseudo-element.

CSS 3 changes pseudo-element syntax to use a double colon (`::`) preceding each pseudo-element. For example, `p::first-letter` refers to the first letter of a paragraph instead of `p:first-letter`. This syntax distinguishes pseudo-elements from pseudo-classes, which use single colon syntax, as in `a:hover`, which is a reference to a pseudo-class.

IE 6 appears to support the double-colon syntax without any problems, but IE 7 does not support this syntax, which is why I present the single colon syntax here. CSS includes more pseudo-elements than those mentioned here; I've selected only those that have the most browser compatibility and support. See Appendix B for additional pseudo-elements.

The following Try It Out shows you what the `::first-letter` and `::first-line` pseudo-elements look like in a style sheet and demonstrates some of the textual effects you can apply.

Try It Out :first-letter and :first-line Pseudo-Elements

Example 3-8. To see how the `:first-letter` and `:first-line` pseudo-elements work, follow these steps.

1. Enter the following markup:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Pseudo-Element Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-8.css' />
  </head>
  <body>
    <p class='quote'>
      You see, wire telegraph is a kind of a very, very long cat.
      You pull his tail in New York and his head is meowing in Los
      Angeles. Do you understand this? And radio operates exactly
      the same way: you send signals here, they receive them there.
      The only difference is that there is no cat.
    </p>
    <p class='byline'>
      - Albert Einstein
    </p>
  </body>
</html>
```

2. Save the preceding markup as `Example_3-8.html`.
3. Enter the following style sheet:

```
p {
  color: darkblue;
  border: 1px solid lightblue;
  padding: 2px;
  font: 14px sans-serif;
}
p.quote:first-letter {
  background: darkblue;
  color: white;
  font: 55px "Monotype Corsiva";
  float: left;

  margin-right: 5px;
}
p.quote:first-line {
  font-weight: bold;
  letter-spacing: 3px;
}
p.byline {
  text-align: right;
  font-style: italic;
  font-size: 10px;
  border: none;
}
```

4. Save the preceding style sheet as `Example_3-8.css`. The markup and CSS that you entered should look something like Figure 3-36 when rendered in a browser.

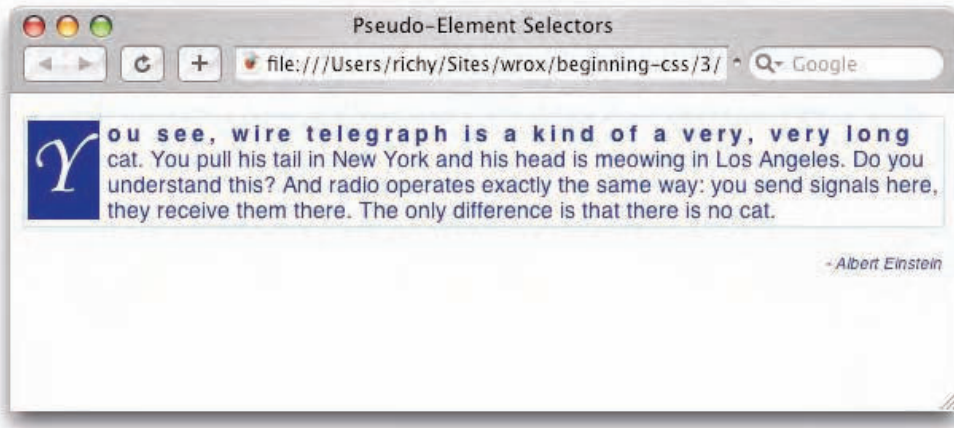


Figure 3-36

How It Works

In Example 3-8 you see an example of both the `:first-letter` and `:first-line` pseudo-elements. The following is a review of the relevant style sheet rules in `Example_3-8.css`. In the following rule, you styled the first letter of the `<p>` element with a class name of `quote`. To select the first letter of the `<p>` element, “Y”, you used the selector `p.quote:first-letter`.

```
p.quote:first-letter {
    background: darkblue;
    color: white;
    font: 55px "Monotype Corsiva";
    float: left;
    margin-right: 5px;
}
```

As shown in Figure 3-36, the first letter of the Einstein quote received a `darkblue` background, `white` text, a large 55-pixel font size, and the font face *Monotype Corsiva*. It’s floated to the left so that subsequent lines wrap around it. It’s given five pixels of right margin.

Then, the first line of the quote receives additional styling. It’s selected with the selector `p.quote::first-line`, and given bold text, in addition to each letter in the line being spaced three pixels apart.

```
p.quote::first-line {
    font-weight: bold;
    letter-spacing: 3px;
}
```

In the next section, I present another type of selector, pseudo-class selectors.

Pseudo-Classes

Pseudo-classes are used to represent dynamic events, a change in state, or a more general condition present in the document that is not easily accomplished through other means. This may be the user's mouse rolling over or clicking on an element. In more general terms, pseudo-classes style a specific state present in the target element. This state may be hovering your mouse cursor over an element, or visiting a hyperlink. Pseudo-classes allow the author the freedom to dictate how the element should appear under either condition. Unlike pseudo-elements, pseudo-classes have a single colon before the pseudo-class property.

Dynamic Pseudo-Classes

The following are considered dynamic pseudo-classes. They are a classification of elements that are only present after certain user actions have or have not occurred:

- ❑ `:link`: signifies unvisited hyperlinks
- ❑ `:visited`: indicates visited hyperlinks
- ❑ `:hover`: signifies an element that currently has the user's mouse pointer hovering over it
- ❑ `:active`: signifies an element on which the user is currently clicking

The first two dynamic pseudo-classes that I discuss are `:link` and `:visited`.

`:link` and `:visited`

The `:link` pseudo-class refers to an unvisited hyperlink, whereas `:visited`, of course, refers to visited hyperlinks. These two pseudo-classes are used to separate styles based on user actions. An unvisited hyperlink may be blue, whereas a visited hyperlink may be purple. Those are the default styles your browser applies. Using dynamic pseudo-classes it is possible to customize those styles.

Figure 3-37 demonstrates how these pseudo-classes are applied.

In Figure 3-37, unvisited links are styled with the `:link` dynamic pseudo-class. They receive `mediumblue` colored text. Visited links, on the other hand have `magenta` colored text.

For obvious reasons, the `:link` and `:visited` pseudo-classes apply only to `<a>` elements.]

```
a:link {  
    color: mediumblue;  
}  
a:visited {  
    color: magenta;  
}
```

Figure 3-37a

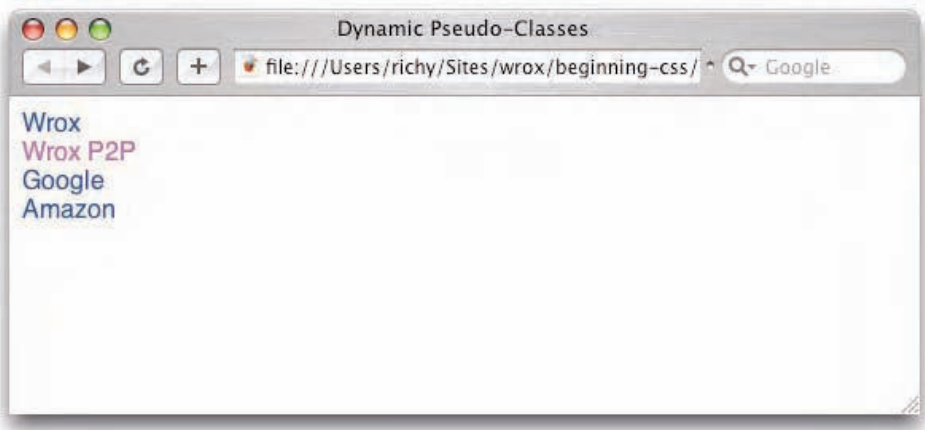


Figure 3-37b

The order in which the `:link` and `:visited` dynamic pseudo-classes appear in the style sheet is important and has to do with the cascade, which I discuss in Chapter 4. If the `:link` pseudo-class is defined after the `:visited` pseudo-class in the style sheet, the `:link` pseudo-class takes precedence. The declarations with the `:link` pseudo-class override those defined for the `:visited` pseudo-class. As you see in Chapter 4, this has to do with how specific the selector is; in this example, the specificity is the same.

A mnemonic device used to remember the order in which dynamic pseudo-classes (as applied to links) must appear in style sheets is *LoVe HAtE*, or `:link`, `:visited`, `:hover` and `:active`.

:hover

The `:hover` pseudo-class refers to an element over which the user's mouse pointer is currently hovering. While the user's mouse pointer is over the element, the specified style is applied; when the user's mouse pointer leaves the element, it returns to the previously specified style. The `:hover` pseudo-class is applied in the same way that the `:link` and `:visited` pseudo-classes are applied. An example of this appears in Figure 3-38.

In Figure 3-38, when the user's mouse hovers over an `<a>` element, the text within the `<a>` element is underlined.

```
a:link {  
    color: mediumblue;  
}  
a:visited {  
    color: magenta;  
}  
a:hover {  
    text-decoration: underline;  
}
```

When the user's mouse cursor hovers over an `<a>` element, the text is underlined.

Figure 3-38a

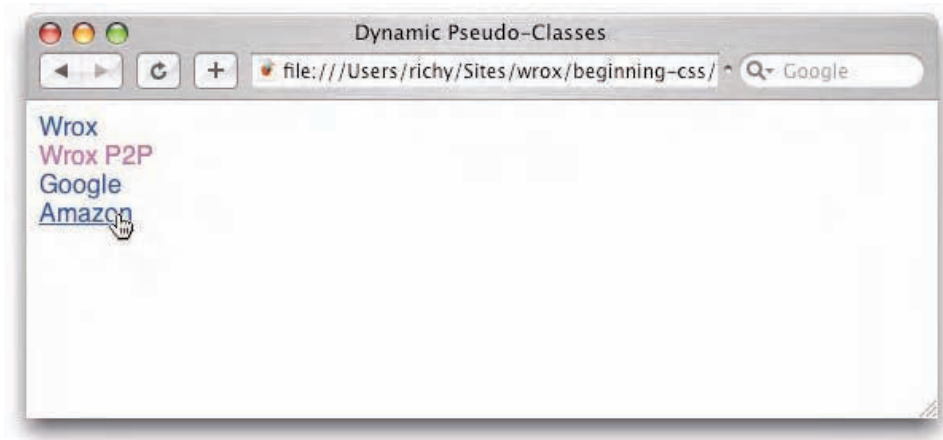


Figure 3-38b

In IE 6, the `:hover` pseudo-class applies only to hyperlinks (which is incorrect under the CSS 2 specification), whereas other browsers recognize the `:hover` pseudo-class on any rendered element, per the CSS 2 specification. This problem is fixed in IE 7.

:active

The `:active` pseudo-class refers to an element that the user is currently clicking and holding down the mouse button on. The specified style remains in place while the user holds down the mouse button, and the element does not return to its original state until the user releases the mouse button. You can see an example of this in Figure 3-39.

In Figure 3-39 you see the `:active` pseudo-class in action. When the user clicks on an `<a>` element, while the mouse button is held down, and before it is released, the element is said to be *active*, in which case the styles in the `:active` pseudo-class rule are applied.

In IE 6 and IE 7, `:active` applies only to hyperlinks; whereas, other browsers allow it to be applied to any element.

```
a:link {
    color: mediumblue;
}
a:visited {
    color: magenta;
}
a:hover {
    text-decoration: underline;
}
a:active {
    color: crimson;
}
```

When the user clicks on an `<a>` element, at the time the click begins to the time the user releases the mouse button, the element is said to be active.

Figure 3-39a

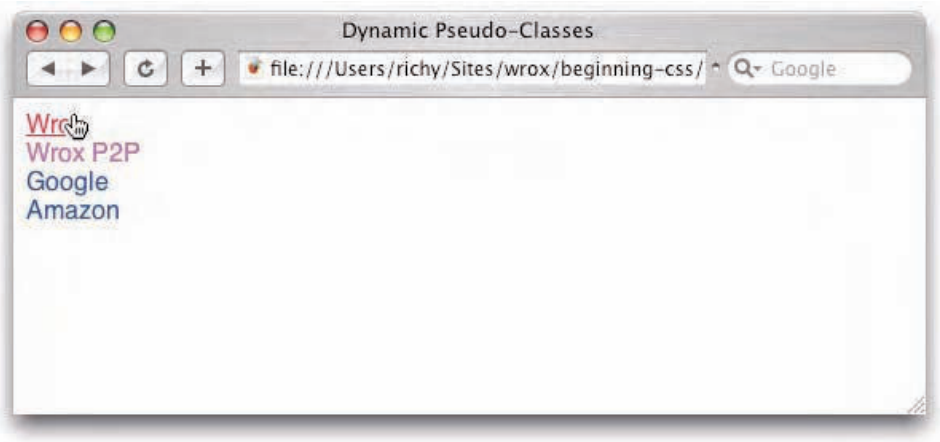


Figure 3-39b

Now that you have been introduced to dynamic pseudo-class selectors, you can try them out for yourself in the following example.

Try It Out Dynamic Pseudo-Class Selectors

Example 3-9. To try out dynamic pseudo-class selectors, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Dynamic Pseudo-Class Selectors</title>
    <link rel='stylesheet' type='text/css' href='Example_3-9.css' />
  </head>
  <body>
    <h1>Proof-of-Concept: Dynamic Pseudo-Class Selectors</h1>
    <ul>
      <li><a href='http://www.wrox.com/'>Wrox</a></li>
      <li><a href='http://p2p.wrox.com/'>Wrox P2P</a></li>
      <li><a href='http://www.google.com/'>Google</a></li>
      <li><a href='http://www.amazon.com/'>Amazon</a></li>
    </ul>
  </body>
</html>
```

2. Save the preceding markup as `Example_3-9.html`.
3. Enter the following CSS into your text editor:

```
body {
  font: 14px sans-serif;
}
h1 {
  font-size: 16px;
}
```

```

ul {
    list-style: none;
}
li {
    margin: 5px 0;
}
a:link {
    color: steelblue;
}
a:visited {
    color: darkorchid;
}
a:hover {
    color: orange;
}
a:active {
    color: crimson;
}

```

4. Save the preceding style sheet as `Example_3-9.css`. Upon completion of the HTML and CSS files, you should see output in your browser like that in Figure 3-40.

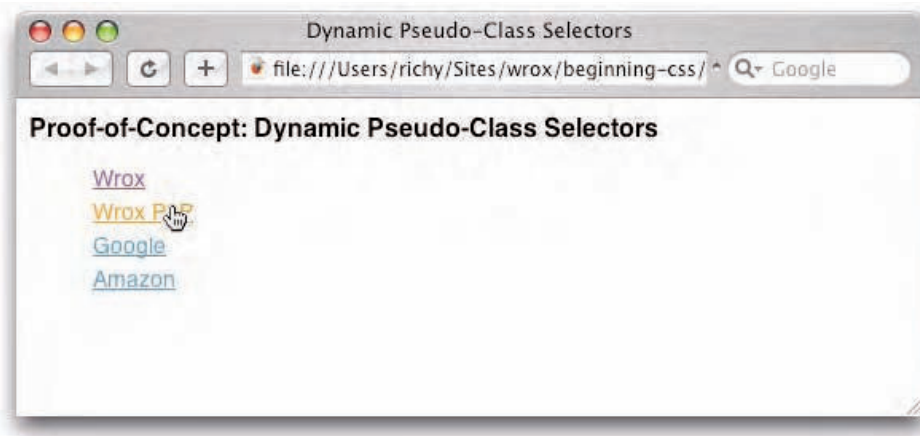


Figure 3-40

How It Works

In Example 3-9, you tried out the dynamic pseudo-classes for yourself. In Example 3-9 there were four dynamic pseudo-classes in use.

The first dynamic pseudo-class that you used styles unvisited links. Unvisited links receive the color `steelblue`.

```

a:link {
    color: steelblue;
}

```

The second dynamic pseudo-class that you used styles visited links. Visited links receive the color `darkorchid`.

Part I: The Basics

```
a:visited {
    color: darkorchid;
}
```

The third selector that you used, the `:hover` dynamic pseudo-class, applies styles when the user's mouse cursor hovers over a link. When a user's mouse cursor comes over a link, the link is colored orange.

```
a:hover {
    color: orange;
}
```

Last, you used the `:active` dynamic pseudo-class, which applies style when the user clicks and holds down the mouse button on a link. When the user clicks and holds down the mouse button, the link is colored crimson.

```
a:active {
    color: crimson;
}
```

The last pseudo-class that I discuss in this chapter is the `:first-child` structural pseudo-class.

The first-child Structural Pseudo-Class

Much like the direct child and next sibling selectors earlier in this chapter, structural pseudo-classes are used to refer to an element's position in a document. The `:first-child` structural pseudo-class applies only when an element is the first child of another element.

IE 6 does not support the `:first-child` structural pseudo-class. See this book's website at www.wrox.com/go/beginning_css2e for compatibility help.

In Figure 3-41, you see an example of the `:first-child` structural pseudo-class. Try it out for yourself in the following example.

```
p:first-child {
    background: moccasin;
    border: 1px solid tan;
}
```

The `:first-child` structural pseudo-class is used to select the first child of another element.

```
<body>
  <p>
    The selector applies to this element, because it is the
    first child.
  </p>
  <p>
    It does not apply to this element, because it is not the
    first child.
  </p>
</body>
</html>
```

Figure 3-41a

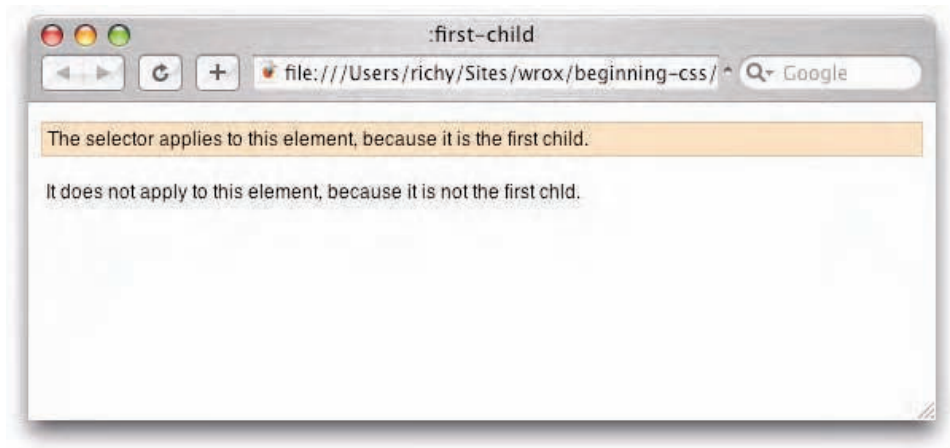


Figure 3-41b

Try It Out The first-child Structural Pseudo-Class

Example 3-10. To see how the `:first-child` structural pseudo-class works, follow these steps.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>:first-child</title>
    <link rel='stylesheet' type='text/css' href='Example_3-10.css' />
  </head>
  <body>
    <h1>Abridged Beatles Discography</h1>
    <table>
      <thead>
        <tr>
          <th>Album</th>
          <th>Year</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Please Please Me</td>
          <td>March 1963</td>
        </tr>
        <tr>
          <td>With The Beatles</td>
          <td>November 1963</td>
        </tr>
        <tr>
          <td>A Hard Day's Night</td>
          <td>July 1964</td>
        </tr>
        <tr>
```

```
        <td>Beatles For Sale</td>
        <td>December 1964</td>
    </tr>
    <tr>
        <td>Help!</td>
        <td>August 1965</td>
    </tr>
    <tr>
        <td>Rubber Soul</td>
        <td>December 1965</td>
    </tr>
    <tr>
        <td>Revolver</td>
        <td>August 1966</td>
    </tr>
    <tr>
        <td>Sgt. Pepper's Lonely Hearts Club Band</td>
        <td>June 1967</td>
    </tr>
    <tr>
        <td>Magical Mystery Tour</td>
        <td>November 1967</td>
    </tr>
    <tr>
        <td>The Beatles (a.k.a. 'The White Album')</td>
        <td>November 1968</td>
    </tr>
    <tr>
        <td>Yellow Submarine</td>
        <td>January 1969</td>
    </tr>
    <tr>
        <td>Abbey Road</td>
        <td>September 1969</td>
    </tr>
    <tr>
        <td>Let It Be</td>
        <td>May 1970</td>
    </tr>
</tbody>
</table>
</body>
</html>
```

2. Save the preceding as `Example_3-10.html`.
3. Enter the following CSS into your text editor:

```
body {
    font-size: 12px sans-serif;
}
table {
    background: slateblue;
    color: #fff;
    width: 100%;
}
```



```

border-collapse: collapse;
border: 1px solid mediumslateblue;
}
td {
border: 1px solid darkslateblue;
padding: 2px;
}
th {
background: lightsteelblue;
color: darkslateblue;
font-size: 18px;
text-align: left;
}
table tbody tr:first-child td {
background: mediumslateblue;
}

```

4. Save the CSS you entered as `Example_3-10.css`. The markup and CSS you entered should look something like what you see in Figure 3-42.



Figure 3-42

How It Works

In Example 3-10, you entered in a table containing some information about albums made by The Beatles. In the style sheet you applied a variety of styles, and among them was an example of the `:first-child` structural pseudo-class.

```

table tbody tr:first-child td {
background: mediumslateblue;
}

```

Part I: The Basics

The preceding rule applies a `mediumslateblue` background to the cells of the first row of the table. It does this because of the `tr:first-child` selector, when `<tr>` is the first child of the `<tbody>` element, which is in turn a descendant of a `<table>` element. The descendant `<td>` elements of the `<tr>` element receive each a `mediumslateblue` background.

Summary

CSS selectors provide a flexible and diverse array of options for applying style to a document. CSS 2 greatly expanded the options made available in CSS 1, with the direct child, attribute value, and next sibling selectors, and CSS 3 has again expanded selector options with selectors like the attribute substring selectors.

In this chapter you learned the following:

- ❑ Selectors may also be user-defined using the `class` and/or `id` attributes.
- ❑ The universal selector applies style to all conceivable page elements.
- ❑ Descendant selectors apply style based on document hierarchy and ancestral relationships.
- ❑ Using child selectors makes the methodology created for descendant selectors more specific.
- ❑ Direct adjacent sibling combinators (that's a mouthful), or as I have termed them, next sibling selectors, apply style if two elements, appearing back to back in a document as siblings, have the same parent.
- ❑ Attribute selectors delegate style depending on the presence of attributes or attribute values.
- ❑ Pseudo-elements are used for situations where it would be difficult to use real markup, such as in the styling of the first letter or first line of a paragraph.
- ❑ Dynamic pseudo-classes are used to style a change in state; examples include visited hyperlinks, rolling the mouse cursor over an element, or actively clicking on an element.

In Chapter 4, I begin discussing concepts also fundamental to CSS, the cascade and inheritance.

Exercises

1. Does the selector `body *` apply to `<input>` elements (assuming an `<input>` element appears between the `<body>` and `</body>` tags)?
2. In the following HTML document, do the selectors `li a` and `li > a` refer to the same element(s)? Can those selectors be used interchangeably? What type of selector is each? Which one is better to use and why?

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
```

```
<title>Dynamic Pseudo-Class Selectors</title>
<link rel='stylesheet' type='text/css' href='Example_3-9.css' />
</head>
<body>
  <h1>Proof-of-Concept: Dynamic Pseudo-Class Selectors</h1>
  <ul>
    <li><a href='http://www.wrox.com/'>Wrox</a></li>
    <li><a href='http://p2p.wrox.com/'>Wrox P2P</a></li>
    <li><a href='http://www.google.com/'>Google</a></li>
    <li><a href='http://www.amazon.com/'>Amazon</a></li>
  </ul>
</body>
</html>
```

3. Given the HTML document in question 2, does the selector `ul + h1` apply? What is the official name of that selector?
4. If you wanted to apply a style based on an HTML attribute's value, what would the selector look like?
5. If you were to style an element based on the presence of an HTML attribute, what would the selector look like?
6. What special character must you include in an attribute value selector to style an element based on what appears at the beginning of an attribute's value? What does a sample selector using that character look like?
7. How many class names can one element have?
8. What special character must you include in an attribute value selector to style an element based on what appears at the end of an attribute's value? What does a sample selector using that character look like?
9. If you wanted to style a link a different color when the user's mouse hovers over it, what might the selector look like?

4

The Cascade and Inheritance

In Chapter 3, I discussed the various types of selectors that CSS supports. In this chapter, now that you have some understanding of the basic nuts and bolts that make up CSS, you continue along that path with the cascade and inheritance. In CSS, inheritance and the cascade are as fundamental as selectors, lengths, and properties. In fact, the importance of precedence is implied by the name of the language itself: cascading style sheets. *Cascading* is a term used to describe precedence. Because CSS declarations can appear more than once for a single element, the CSS specification includes a set of guidelines defining which declarations can take precedence over others and how this is decided. In this chapter, I discuss the following:

- ☐ The *cascade* and how style sheets and some selectors take precedence over others
- ☐ Inheritance and why the values of some properties are inherited and some are not
- ☐ The `!important` rule and how to force precedence
- ☐ Custom style sheets and how to override website styles with them

The Cascade

Style sheets can come from more than one place. A style sheet can originate from any of the following sources:

- ☐ From the browser (default look and feel)
- ☐ From the user visiting the website (a user-defined style sheet)
- ☐ From the web page itself (the website's author)

Because a style sheet can originate from more than one source, it is necessary to establish an order of precedence to determine which style sheet applies style for the page the user is seeing. The first style sheet comes from the browser, and this style sheet applies some default styles for a web page, such as the default font and text color, how much space is applied between each line of text, and how much space is applied between each letter of text. In a nutshell, it controls the look and feel of the web page by controlling the behavior of each element when no styles are specified.

Part I: The Basics

A style sheet can also be applied by a user visiting the website via a user-defined style sheet, which is discussed later in this chapter. This allows the user to specify his or her own look and feel. This aspect of CSS makes the web more accessible: A user with visual disabilities can write a style sheet to accommodate his or her needs, or the browser can provide options that generate the user's style sheet behind the scenes. No knowledge of CSS is required.

Finally, the author of the web page can specify a style sheet (of course). The precedence of each style sheet is as follows:

- ❑ The browser's style sheet is the weakest.
- ❑ The user's style sheet takes precedence over the browser's style sheet.
- ❑ The author's style sheet is the strongest and takes precedence over the user's and the browser's style sheets.

The (X)HTML `style` attribute is more important than styles defined in any style sheet.

You might be wondering what kind of styles does the browser apply? Figure 4-1a demonstrates this.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Default Styles</title>
  </head>
  <body>
    <h1>Default Styles</h1>
    <p>
      Browsers apply default styles to some elements.
    </p>
    <p>
      Examples include:
    </p>
    <ul>
      <li>
        Margin or padding is applied to the <body> element.
      </li>
      <li>
        Margin is applied to heading elements <h1> through
        <h6>;
      </li>
      <li>Margin is applied to <p> elements.</li>
      <li>Margin or padding is applied to the <ul> element.</li>
    </ul>
  </body>
</html>
```

Figure 4-1a

This results in the output in Figure 4-1b.

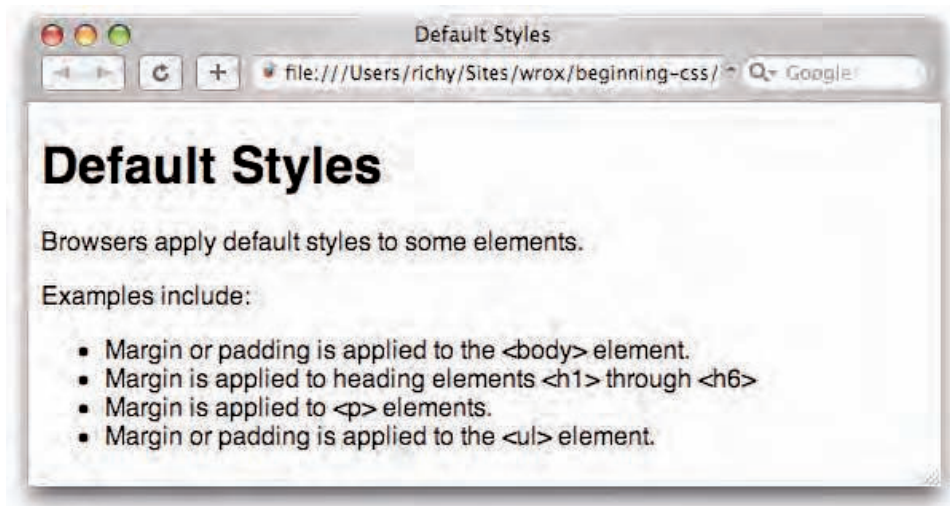


Figure 4-1b

In Figure 4-1b, you can see an example of some of the default styles that a browser applies. One example is the spacing between the heading “Default Styles” and the text in the paragraph that follows. The unordered list (element) has a bullet before each list item (the element).

Figure 4-2a demonstrates a style sheet that removes the default styles shown in Figure 4-1b.

```
body {  
    margin: 0;  
    padding: 0;  
}  
h1 {  
    margin: 0;  
    font-weight: normal;  
    font-size: 16px;  
}  
p {  
    margin: 0;  
}  
ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
}
```

On the <body> element, some browsers apply margin, while others apply padding. I talk about the difference between these two properties in Chapter 7.

Figure 4-2a

The style sheet in Figure 4-2a is applied to the markup in Figure 4-1a, which results in the output in Figure 4-2b.

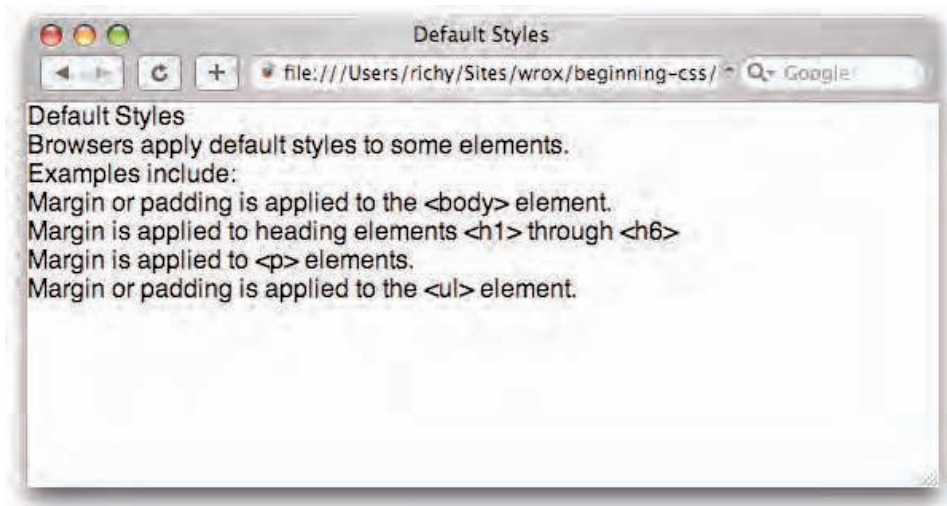


Figure 4-2b

When you compare Figure 4-2b with Figure 4-1b, you get an idea of what kinds of styles a browser applies by default. The browser applies spacing between elements and depending on the element, that spacing can be controlled by either the `margin` or the `padding` property. You learn more about those two properties in Chapter 7, "The Box Model." Figure 4-2 demonstrates, however, that it is possible to override the browser's default styles. Overriding the default styles is made possible by the cascade.

The cascade sets the order of precedence, and in Figure 4-2, it says that my style sheet rules (the author's) have stronger precedence (are more important) than the browser's built-in style sheet rules.

By and large, there are only two situations that a web designer will ever encounter in composing a style sheet: overriding the browser's default styles, and overriding styles set in other style sheets within the same website, that is, overriding the web designer's own styles set elsewhere in the same document.

In CSS, the precedence is determined by how specific a selector is. That is to say a vague selector has less precedence than a more specific selector. In the next section, I discuss how to find out how specific a selector is using a simple, easy-to-remember formula.

Calculating the Specificity of a Selector

In addition to style sheet precedence, an order of precedence exists for the selectors contained in each style sheet. This precedence is determined by how specific the selector is. For instance, an `id` selector is the most specific, and the universal selector is the most general. Between these, the specificity of a selector is calculated using the following formula:

- ❑ Count 1 if the styles are applied from the (X)HTML style attribute, and 0 otherwise; this becomes variable *a*.
- ❑ Count the number of ID attributes in the selector; the sum is variable *b*.

- ❑ Count the number of attributes, pseudo-classes, and class names in a selector; the sum is variable *c*.
- ❑ Count the number of element names in the selector; this is variable *d*.
- ❑ Ignore pseudo-elements.

Concatenate each number together to get the specificity of the selector. *Concatenate* is a programming term that means *glue together*. In this case if I concatenate *a*, *b*, *c*, and *d* I get *abcd*, instead of the sum of *a*, *b*, *c*, and *d*, which I might refer to as *e*. Following are some examples.

Selector	Selector Type	Specificity
*	Universal Selector	0000 (a = 0, b = 0, c = 0, d = 0)
li	Element Name	0001 (a = 0, b = 0, c = 0, d = 1)
ul li	Element Name	0002 (a = 0, b = 0, c = 0, d = 2)
div h1 + p	Element Name	0003 (a = 0, b = 0, c = 0, d = 3)
input[type='text']	Element Name + Attribute	0011 (a = 0, b = 0, c = 1, d = 1)
.someclass	Class Name	0010 (a = 0, b = 0, c = 1, d = 0)
div.someclass	Element Name + Class Name	0011 (a = 0, b = 0, c = 1, d = 1)
div.someclass.someother	Element Name + Class Name + Class Name	0021 (a = 0, b = 0, c = 2, d = 1)
#someid	ID Name	0100 (a = 0, b = 1, c = 0, d = 0)
div#someid	Element Name + ID Name	0101 (a = 0, b = 1, c = 0, d = 1)
style (attribute)	style (attribute)	1000 (a = 1, b = 0, c = 0, d = 0)

I have included the leading zeros in the specificity chart to clarify how concatenation works, but these are actually dropped. To determine the order of precedence, simply determine the highest number. The selector with the highest number wins. Consider the example in Figure 4-3a.

```
body {  
    font-size: 24px;  
}  
p#none {  
    background: none;  
}  
p {  
    background: yellow;  
}
```

The <p> element with id name none, doesn't get a yellow background, because its specificity is 101, which is much higher than the general <p> element selector's specificity of 1.

Figure 4-3a

Apply the CSS in Figure 4-3a to the markup in Figure 4-3b.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>  
    <head>  
        <title>Specificity</title>  
        <link rel='stylesheet' type='text/css' href='096977%20fg0403.css' />  
    </head>  
    <body>  
        <p>  
            This paragraph has a yellow background.  
        </p>  
        <p id='none'>  
            This paragraph doesn't have a yellow background,  
            because the id selector is more specific than the  
            element selector.  
        </p>  
    </body>  
</html>
```

Figure 4-3b

The result looks like the output shown in Figure 4-3c.

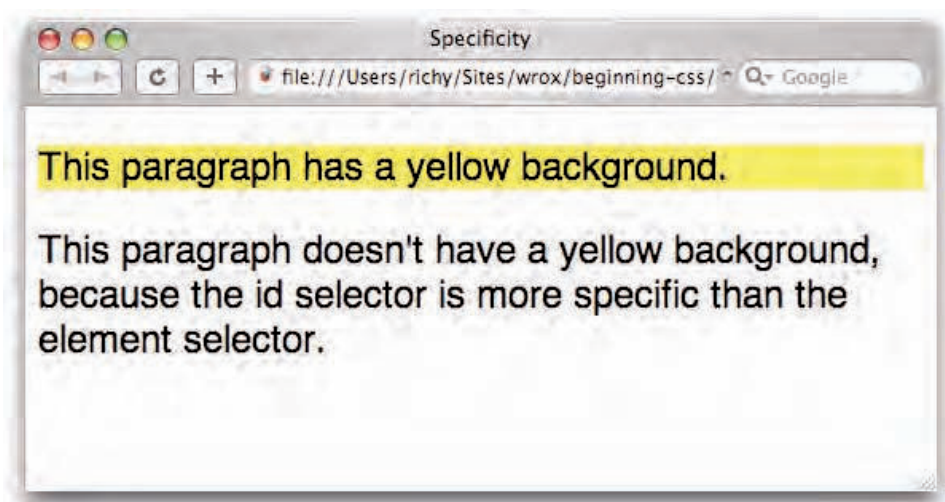


Figure 4-3c

In Figure 4-3, you see an example of precedence via the selector's specificity. In Figure 4-3a, a rule is set for all `<p>` elements to have a yellow background. Because the selector is vague, it has a low specificity. Using the table at the beginning of this section, you find that the selector

```
p {  
    background: yellow;  
}
```

has a specificity of 1, which is very low. The `<p>` element with id name `none` is set to have no background, and because it has a higher specificity than the other selector, which again using the table at the beginning of this section you find the specificity to be 101, results in the `<p>` element with id name `none` having no background.

In Figure 4-3, you can see that the order that the rules appeared in the style sheet does not matter; the rules can swap places in the style sheet and the outcome would be the same. So you might be asking yourself, does the order matter? Sometimes, it does matter, but only when there are two rules of the same specificity. Consider the example in Figure 4-4a.

```
body {  
    font-size: 24px;  
}  
p {  
    background: none;  
}  
p {  
    background: yellow;  
}
```

If two or more selectors have the same specificity, the last one wins.

Figure 4-4a

Apply the style sheet in Figure 4-4a to the markup in Figure 4-4b.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>  
    <head>  
        <title>Specificity</title>  
        <link rel='stylesheet' type='text/css' href='096977%20fg0404.css' />  
    </head>  
    <body>  
        <p>  
            This paragraph has a yellow background.  
        </p>  
        <p>  
            This paragraph also has a yellow background.  
        </p>  
    </body>  
</html>
```

Figure 4-4b

The result is shown in Figure 4-4c.

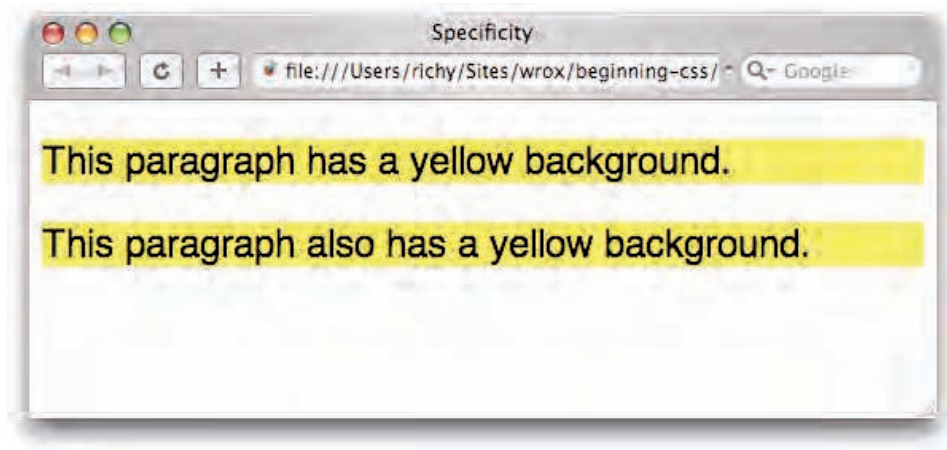


Figure 4-4c

In Figure 4-4c, you see that when two or more selectors have the same specificity, the last one wins.

When an (X)HTML `style` attribute is applied, it is considered the most specific of any selector on the page. That's because according to the CSS specification, it is defined as having a specificity all of its own, that is higher than any other. The `style` attribute has a specificity of 1000. Because the `style` attribute appears after any styles appearing in style sheets, it also takes precedence over the all other selectors. Therefore, the `style` attribute takes precedence over all other rules.

Try It Out Experimenting with Specificity

Example 4-1. Follow these steps to experiment with specificity.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Specificity</title>
    <link rel='stylesheet' type='text/css' href='Example_4-1.css' />
  </head>
  <body>
    <p>
      Specificity is determined by how specific the selector is.
      <span id='specific'>A specific selector wins</span>
      over a <span>more general one</span>.
    </p>
    <p>
      Order isn't important until there are one or more elements
      of the same specificity referring to the same element. In
      which case, <span>the last one wins</span>.
    </p>
  </body>
</html>
```

2. Save the preceding document as `Example_4-1.html`.

3. Enter the following CSS into your text editor:

```
body {  
    font: 14px sans-serif;  
}  
span#specific {  
    background: pink;  
}  
span {  
    background: red;  
}  
span {  
    background: yellow;  
}
```

4. Save the preceding style sheet as `Example_4-1.css`. Example 4-1 results in the output you see in Figure 4-5.

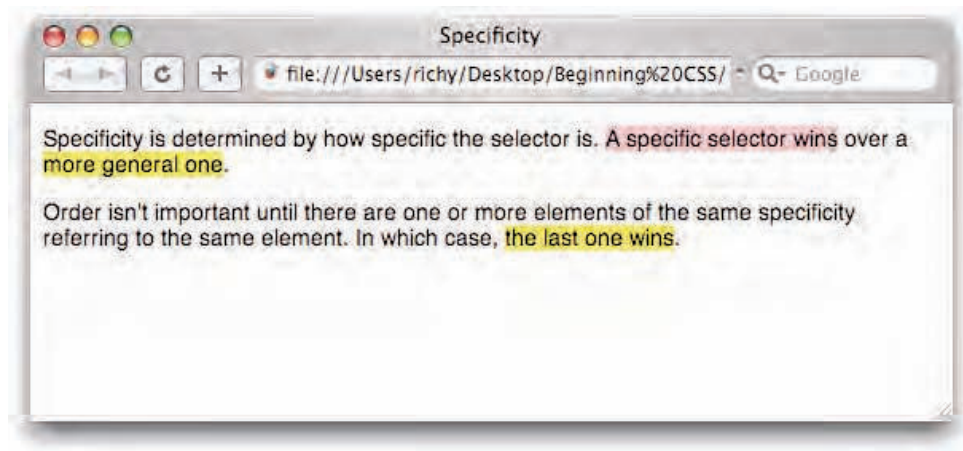


Figure 4-5

How It Works

In Example 4-1, you saw an example of the cascade in action. In the markup there are three `` elements, and one has an id name of `specific`. It gets a pink background because the selector `span#specific` has a specificity of 101, which is more specific than the subsequent selectors, which each have a specificity of 1.

```
span#specific {  
    background: pink;  
}
```

Then there are two additional rules in the style sheet, each with the same specificity of 1. The last selector wins, since both selectors have the same specificity of 1, which in turn results in the last two `` elements in the markup getting yellow backgrounds.

```
span {  
    background: red;  
}
```

Part I: The Basics

```
span {  
    background: yellow;  
}
```

In the next section, I describe how you can override specificity by including special syntax within a CSS declaration.

!important Rules

Along with the need for the cascade in CSS came the need to override it. This is where `!important` rules come in. The `!important` syntax appears within a declaration, after the property value and before the semicolon that terminates the declaration. Two components make up this syntax: an exclamation mark, used here as a delimiter, and the `important` keyword. A *delimiter* marks the ending of one thing and the beginning of another. Here the exclamation mark signals the end of the declaration. The `important` keyword must appear next, followed by a semicolon to terminate the declaration; this is demonstrated in Figure 4-6a.

```
body {  
    font-size: 24px;  
}  
p {  
    background: lightblue !important;  
}  
p {  
    background: none;  
}
```

The `!important` rule takes precedence.

Figure 4-6a

A declaration containing the `!important` rule, like the preceding one, takes precedence over any other declaration. The CSS in Figure 4-6a is combined with the markup in Figure 4-6b.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>  
    <head>  
        <title>Specificity, !important</title>  
        <link rel='stylesheet' type='text/css' href='096977%20fg0406.css' />  
    </head>  
    <body>  
        <p>  
            This paragraph has a lightblue background.  
        </p>  
        <p>  
            This paragraph also has a lightblue background.  
        </p>  
    </body>  
</html>
```

Figure 4-6b

The result of Figure 4-6a and Figure 4-6b result in the output in Figure 4-6c.

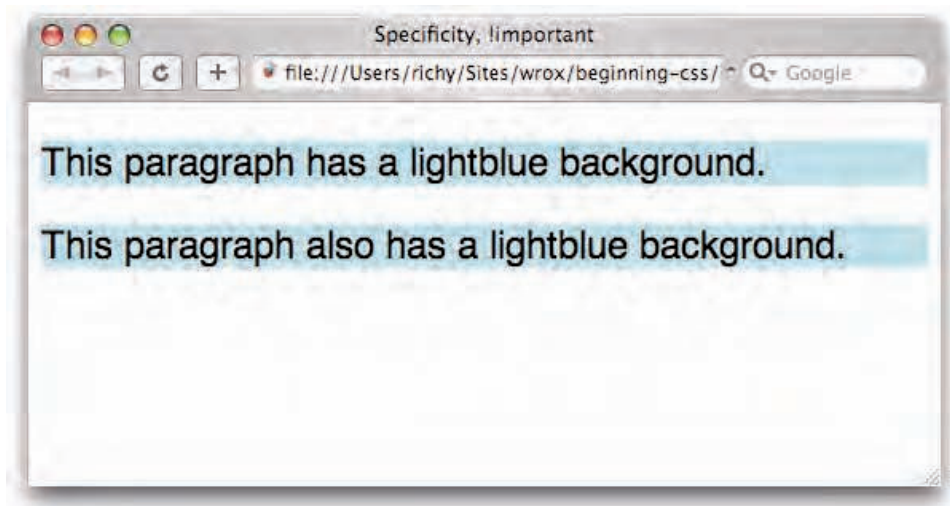


Figure 4-6c

In Figure 4-6, you see the same example as you saw in Figure 4-4—two selectors for `<p>` elements with different background declarations. In Figure 4-4, the last selector won because both selectors have the same specificity. In Figure 4-6, the first selector includes the `!important` syntax, which causes the cascade to be overridden, and thus makes the background of both `<p>` elements in the XHTML document lightblue.

The `!important` rule also takes precedence over the style attribute. Figure 4-7 is an example of this.

```
body {
    font: 24px sans-serif;
}
p {
    background: pink !important;
}
```

The `!important` rule takes precedence over the (X)HTML style attribute.

Figure 4-7a

The CSS in Figure 4-7a is combined with the markup in Figure 4-7b.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Specificity, !important</title>
    <link rel='stylesheet' type='text/css' href='096977%20fg0407.css' />
  </head>
  <body>
    <p>
      This paragraph has a pink background.
    </p>
    <p style='background: lightblue;'>
      This paragraph also has a pink background.
    </p>
  </body>
</html>
```

Figure 4-7b

Part I: The Basics

The CSS in Figure 4-7a and the markup in Figure 4-7b result in the output in Figure 4-7c.

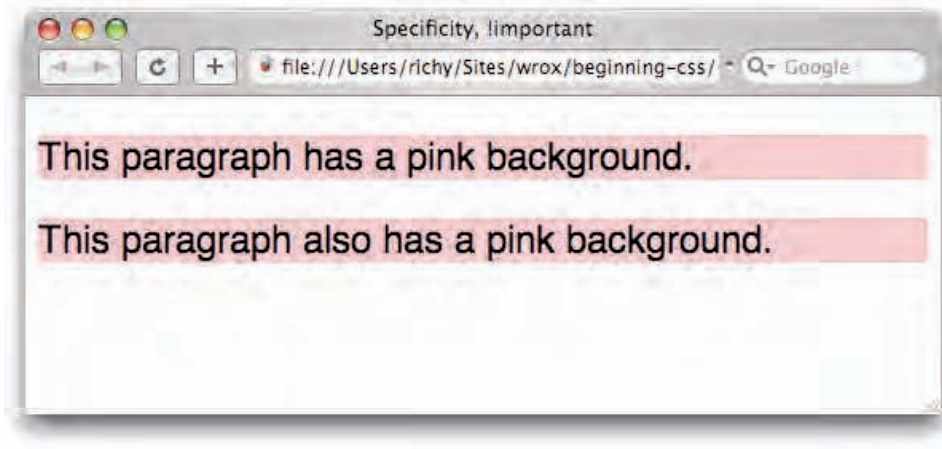


Figure 4-7c

In Figure 4-7c, you see that the background for both paragraphs is pink, despite one of the two paragraphs having a declaration setting the background of that `<p>` element to `lightblue`, which demonstrates to you that the `!important` rule takes precedence over even the `style` attribute.

If more than one `!important` rule appears in a style sheet, and the style sheet has the same origin — that is, both rules come from the author's style sheet or both come from the user's style sheet — the latter rule wins out over any specified previously.

Try It Out Working with !important Rules

Example 4-2. Follow these steps to experiment with specificity.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Specificity, !important</title>
    <link rel='stylesheet' type='text/css' href='Example_4-2.css' />
  </head>
  <body>
    <p>
      !important rules are used to override specificity. The
      !important syntax causes a selector to have
      <span id='precedence'>
        greater precedence than those without it.
      </span>
      It also
      <span style='background: lightblue'>
        has greater precedence than the (x)HTML style attribute.
      </span>
    </p>
  </body>
</html>
```

2. Save the preceding document as `Example_4-2.html`.
3. Enter the following CSS into your text editor:

```
body {  
    font: 14px sans-serif;  
}  
span#precedence {  
    background: lightyellow;  
}  
span {  
    background: orange !important;  
}
```

4. Save the preceding style sheet as `Example_4-2.css`. Example 4-2 results in the output shown in Figure 4-8.

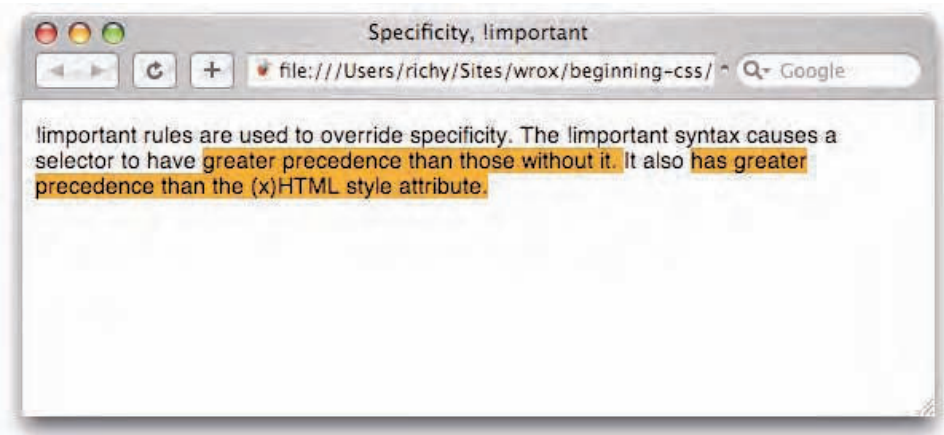


Figure 4-8

How It Works

In Example 4-2, you see how the `!important` rule overrides precedence. Because the following declaration contains the `!important` syntax, it causes the background of all the `` elements to be orange.

```
span {  
    background: orange !important;  
}
```

So far you've seen precedence, a concept that decides how the browser applies styles based on the importance of the selector. In the next section, I talk about inheritance, which is how the browser applies certain styles to an element and all that element's children.

Inheritance

CSS is designed to simplify web document creation, enabling a property to be applied to all elements in a document. To put it another way, after a property has been applied to a particular element, its children retain those property values as well. This behavior is called *inheritance*.

Part I: The Basics

Many properties in CSS are inheritable; some are not. Where it is supported and appropriate, inheritance makes writing style sheets a snap. For the most part, two types of properties can be inherited: text and font properties. Figure 4-9 shows an example of inheritance.

```
body {  
    font-size: 24px;  
}  
div {  
    color: crimson;  
    text-align: right;  
    border: 1px solid crimson;  
    padding: 10px;  
}
```

Properties that are inheritable, such as the color or the text-align properties, are inherited by all of the element's children.

Figure 4-9a

The CSS in Figure 4-9a is combined with the markup in Figure 4-9b.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>  
    <head>  
        <title>Inheritance</title>  
        <link rel='stylesheet' type='text/css' href='096977%20fg0409.css' />  
    </head>  
    <body>  
        <div>  
            <h1>Inheritance</h1>  
            <p>  
                Some properties in CSS are inherited to children elements  
                as you can see here. The <h1> heading and the  
                <p> element inherit color and alignment from the  
                <div>, but not the border and the padding.  
            </p>  
        </div>  
    </body>  
</html>
```

Figure 4-9b

The CSS in Figure 4-9a and the markup in Figure 4-9b result in the output in Figure 4-9c.

In the preceding code, the rule is applied to the <div> element, and the color and text-align properties are inherited by the <h1> and <p> elements contained within the <div> element. The advantage of inherited properties is that you don't have to specify a property again for each nested element. On the other hand, the border and the padding properties are not inherited, since it is not likely a web designer would desire those properties to be inherited. Figure 4-10 shows what Figure 4-9 would look like if the border and padding properties were inherited.

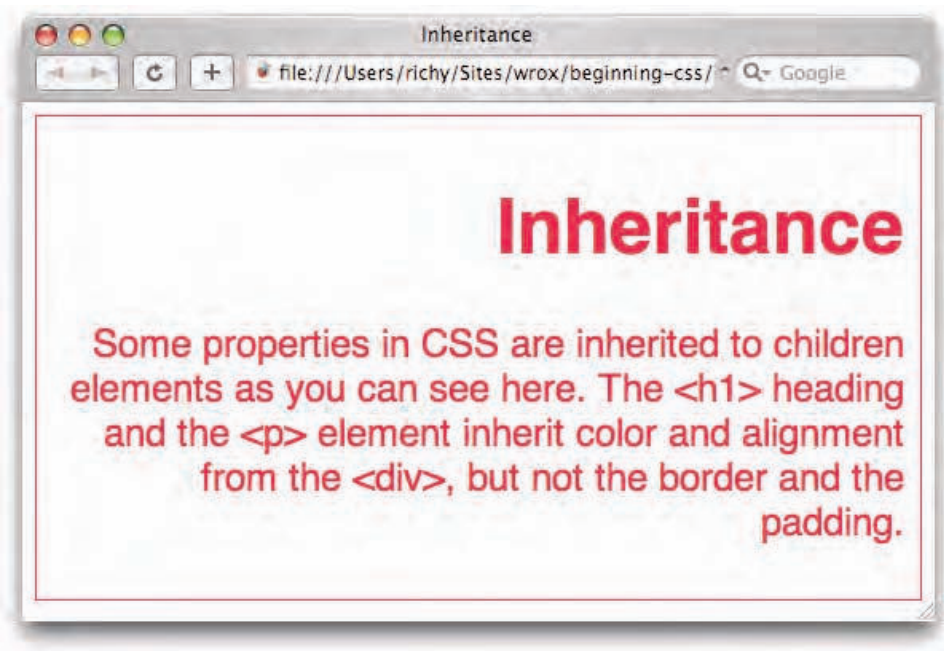


Figure 4-9c

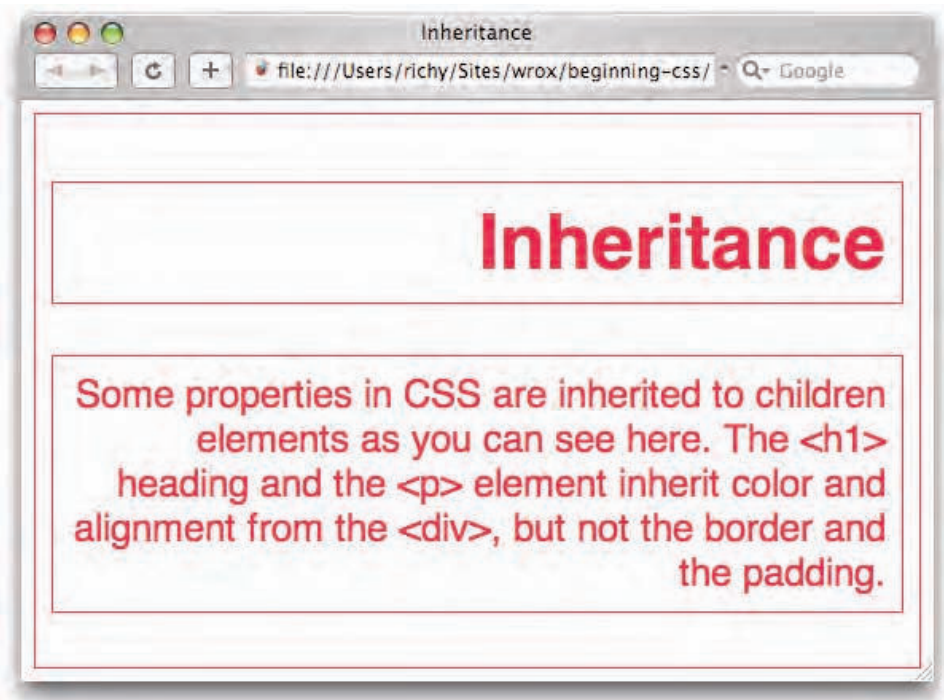


Figure 4-10

Part I: The Basics

In Figure 4-10, you see that some properties, such as `border` and `padding`, are not inherited because inheriting would not be appropriate. Most of the time, you want these to be set only on a selected element and not on that selected element's children elements. I discuss the `border` and `padding` properties in more detail in Chapter 7, "The Box Model."

Inheritance for each property is outlined in Appendix B.

Try It Out Working with Inheritance

Example 4-3. Follow these steps to experiment with inheritance.

1. Enter the following markup into your text editor:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en'>
  <head>
    <title>Inheritance</title>
    <link rel='stylesheet' type='text/css' href='Example_4-3.css' />
  </head>
  <body>
    <p>
      In CSS, some properties are inherited, such as the color, font,
      and text properties. Other properties, such as border, margin,
      and padding, are not inherited, since it wouldn't be
      practical.
    </p>
  </body>
</html>
```

2. Save the preceding document as `Example_4-3.html`.
3. Enter the following CSS into your text editor:

```
body {
  font: 14px sans-serif;
  color: darkslateblue;
  border: 5px dashed darkslateblue;
  margin: 10px;
  padding: 10px;
  text-align: center;
}
```

4. Save the preceding style sheet as `Example_4-3.css`. Example 4-3 results in the output shown in Figure 4-11.

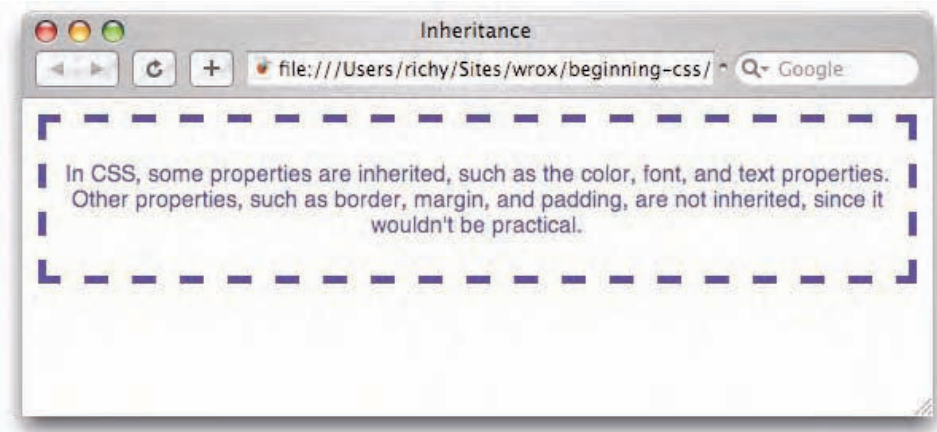


Figure 4-11

How It Works

In Example 4-3, you see an example of inheritance. In the style sheet, the properties `font`, `color`, and `text-align` are inherited by the `<p>` element, while the `border`, `margin`, and `padding` properties are not inherited.

```
body {  
    font: 14px sans-serif;  
    color: darkslateblue;  
    border: 5px dashed darkslateblue;  
    margin: 10px;  
    padding: 10px;  
    text-align: center;  
}
```

Summary

Inheritance and the cascade are fundamental to CSS. Inheritance makes controlling the effects of property values a breeze, because each property is defined either to inherit or not, as is appropriate to its purpose. The cascade provides some rules for precedence to determine which styles win when multiple style sheets and rules containing the same declarations come into play. Precedence is determined by a simple formula that calculates which selector wins. In this chapter you learned the following:

- ❑ Some properties are inherited, which reduces redundancy in the document by eliminating the need for declarations to be written multiple times.
- ❑ Some properties are not inherited, which also reduces redundancy by preventing the effects of declarations from being applied to the element's descendants.
- ❑ The cascade provides both some ground rules and a simple formula to determine the precedence of style sheets and selectors.

Part I: The Basics

Now that you know the background of CSS, Chapter 5 introduces you to CSS's text manipulation properties.

Exercises

1. In the following style sheet, determine the specificity of each selector.

```
ul#hmenu ul.menu {
    margin: 0;
    padding: 0;
    list-style: none;
    position: absolute;
    top: 35px;
    left: 0;
    width: 100%;
    visibility: hidden;
    text-align: left;
    background: rgb(242, 242, 242);
    border: 1px solid rgb(178, 178, 178);
    border-right: 1px solid rgb(128, 128, 128);
    border-bottom: 1px solid rgb(128, 128, 128);
}
ul#hmenu li li:hover {
    background: rgb(200, 200, 200);
}
ul#hmenu ul.menu ul.menu {
    top: -1px;
    left: 100%;
}
ul#hmenu li#menu-204 ul.menu ul.menu,
ul#hmenu li#menu-848 ul.menu ul.menu ul.menu ul.menu,
ul#hmenu li#menu-990 ul.menu ul.menu {
    left: auto;
    right: 100%;
}
ul#hmenu > li.menu.eas + li.menu.eas ul.menu ul.menu ul.menu ul.menu {
    right: auto;
    left: 100%;
}
li.menu,
li.menu-highlight {
    position: relative;
}
ul.menu li a {
    text-decoration: none;
    color: black;
    font-size: 12px;
    display: block;
    width: 100%;
    height: 100%;
}
ul.menu li a span {
    display: block;
```



```
padding: 3px 10px;
}
ul.menu span.arrow {
    position: absolute;
    top: 2px;
    right: 10px;
    width: 11px;
    height: 11px;
    background: url('/images/arrow.gif') no-repeat;
}
```

2. According to the following style sheet, what color is the link?

```
a.context:link {
    color: blue;
}
a.context:visited {
    color: purple;
}
a.context:hover {
    color: green;
}
a.context:active {
    color: red;
}
```

3. According to the following style sheet, what color is the link?

```
a.context:visited {
    color: purple;
}
a.context:hover {
    color: green;
}
a.context:active {
    color: red;
}
a.context:link {
    color: blue;
}
```

4. According to the following style sheet, what color is the link?

```
a.context:link {
    color: blue;
}
a.context:visited {
    color: purple !important;
}
a.context:hover {
    color: green;
}
a.context:active {
    color: red;
}
```